```
public abstract class Node {
   private Node[] children;

   public Node( Node[] children ){
      this.children = children;
   }
   public int getNumChildren(){
      return children.length;
   }

   public Node getChild( int index ){
      return children[ index ];
   }
}

public class IntNode extends Node {
   private int value;

   public IntNode( Node[] children, int value ){
      super( children );
      this.value = value;
   }

   public int getValue(){
      return value;
   }
}
```

Appendix B

마이크로소프트 인턴 면접기

2007년 여름방학 동안에 마이크로소프트 인턴쉽을 위해 겪은 면접 경험담을 써보고자 한다. 이 이야기는 어디까지 나의 경험과 역시 마이크로소프트 인턴 면접을 본 주변 친구들의 이야기를 기반으로 쓰여진 것이므로 회사마다, 시기마다, 학교마다 얼마든지 이 내용은 바뀔 수 있다. 간단히 나를 소개하면, 한국에서 학부를 마치고 산업기능요원으로 짧지 않은 기간 동안 소프트웨어를 개발하였다. 그리고 2006년 가을 학기부터 미국의 모 대학에서 전산학 석사 과정에 있다. 먼저, 일반적인 미국 대학생들의 인턴쉽 프로그램에 대해 이야기를 해보고 구체적인 마이크로소프트 면접 이야기를 해보자. 그리고 마지막으로는 나와 친구들이 면접 때 받은 코딩 문제들을 살펴보자.

미국 대학생들을 위한 인턴쉽 프로그램

미국 대학의 학기가 우리와 다르다는 사실은 잘 알 것이다. 보통 8월말에서 12월초까지 가을 학기, 그리고 1월초에서 5월초까지 봄 학기로 크게 나뉜다(예외로 캘리포니아 주에 있는 대학교들은 쿼터(Quarter) 제도로 움직이기도 한다). 그리고 여름 방학은 보통 5~8월 동안 약 3~4개월 정도 지속된다. 그래서 이런 긴 여름 방학 동안 많은 업체들이 대학생들을 위한 인턴쉽(이하 인턴으로도 혼용하여 표현) 프로그램을 제공하고 있고, 학생들도 이 기회를 적극적으로 이용한다. 보통 인턴은 12~15주 정도 지속이 되어 결코 짧지 않는 기간 동안 일을 할 수 있다. 단순 업무 보조가 아닌 실제 프로젝트를 수행하는 경우가 많다.

반면, 우리나라의 대학들은 여름 방학과 겨울 방학이 각각 두 달 정도로 균등히 배분이 되어있다. 그래서 제대로 된 인턴을 하기가 힘들다. 또한, 이런 짧은 여름 방학으로 인하여 외국 기업의 인턴으로 가는 것도 현실적으로 힘들다. 카이스트 같은 경우에는 이제 여름 방학을 미국처럼 길게 하여 학생들에게 보다 많은 인턴 기회를 주려고 하고 있다. 우리 대학들도 좀 더 내실 있는 인턴 경험을 할 수 있도록 학기를 조정할 필요가 있다.

인턴을 지원하는 학생이 학부생이냐, 석사 과정이냐, 그리고 박사 과정이냐에 따라 그 면접 과정이 사뭇 다를 수 있다. 인턴 자리는 크게 일반적인 개발직과 연구직으로 나눌 수 있다. 대개 박사 과정 학생들은 교수님들의 추천을 받아 연구직으로 가는 경우가 많다. 그리고 이 경우에는 알고리즘 문제보다는 지금 이 학생이 연구하고 있는 주제에 대한 질문이 이어지는 경우가 많다.

반면, 석사 과정이나 학부 3~4학년 학생들은 주로 개발자로 인턴을 가게 된다. 어떤 회사는 특정 플랫폼의 개발 경험이 얼마나 있느냐, 특정 분야에 대한 지식이 얼마나 있느냐를 물을 수 있다. 그러나 대개 그것보다는 일반적인 전산 능력 즉, 알고리즘과 프로그래밍 테스트를 하는 경우가 많다. 대표적으로 마이크로소프트, 구글, 그리고 아마존 같은 기업이 알고리즘을 묻는 방식으로 진행된다. 전형적인 면접 진행 방식은 책에서도 잘 소개가 되어있지만, 비교적 간단한 1차 인터뷰, 그리고 힘들고 긴 2차 인터뷰를 거치는 경우가 보통이다.

그렇다면 어떻게 인턴으로 지원할 수 있을까? 나의 경우에는 2월 중순에 학교에서 열린 ECE/CS(전기/컴퓨터/전산) 학과를 위한 Job Fair를 이용하여 지원을 하였다. 인텔, MS, VMware, 아마존과 같은 큰 업체를 비롯해 많은 컴퓨터 관련 업체들이 와서 이력서를 받는다. 그 외에도 하드웨어 회사인 AMD, Broadcom 등의 회사도 볼 수 있었다.

특히 그 자리에서 바로 간단한 프로그래밍 관련 질문을 하는 곳도 있다. 예를 들어, 아마존 같은 경우에는 간단히 volatile이라는 C언어 키워드가 무엇을 하는지 묻기도 했고, C++ 클래스 상속에 대한 질문을 묻기도 하였다. 나는 윈도우 프로그래밍을 긴 시간 동안 하였고, 때마침 지난 학기에 윈도우 플랫폼에서 메모리 버퍼 오버플로를 막는 프로젝트를 한 것이 있어서 MS에 관심이 많았다. MS 부스에는 이미 많은 학생들이 줄을 서서 자신을 광고하고 있었다. 드디어 내 차례가 되었다. 리쿠르터랑 지금까지 내가 한 것에 대해서 간략하게 말을 하면서 이야기를 주고받았다. 그리고 이력서를 던져 준다. 던져 준다는 표현이 정확할 정도로 많은 이력서가 쌓인다.

그 외에는 Monster Track[1]이라는 일종의 리쿠르팅 사이트를 통해 구하는 방법도 있다. 아니면 MS의 경우에는 자신이 다니는 학교의 리쿠르터에게 직접 레주메를 보내 연결할 수도 있다. 대략 50~60개의 학교가 등록[2]되어있고 각 학교마다 리쿠르터가 배정되어있다.

그리고 하나 느낀 것은 MS의 여름 인턴을 가기 위해서는 반드시 지난 해 가을 학기부터 지원을 하여야 한다는 것이다. 그래야 일찍(1~2월 사이에) 최종 면접까지 보고 확답을 얻을 수 있다. 나 같은 경우에는 일정이 굉장히 미뤄져서 학기가 끝나고 나서야 겨우 시애틀을 갈 수 있을 정도였다. 중간마다 일정을 조율하는 리쿠르터가 계속 바뀌는데, 중간의 한 담당자와 3주간 연락이 두절되어 꽤 늦어졌다. 또한 마이크로소프트 인턴 지원은 자기가 특정 팀을 고를 수 있는 것이 아니고, 1차 면접관이 팀을 배정한다. 그래서 이미 좋은 팀은 모두 마감이 될 수도 있다.

이번에는 MS 인턴쉽에서 어떤 자리를 뽑는지 이야기를 해보자. 크게 Program Manager(PM), Software Development Engineer(SDE), Software Development Engineer in Testing(SDET[3]), 그리고 이 글을 쓰는 (2007년 8월) 현재, Hardware Engineer라는 포지션도 추가가 되어 총 4개 정도의 직군이 있다. 그리고 MS가 아닌 Microsoft Research에서도 연구 과정을 위한 인턴쉽을 뽑는다. 이 과정에서도 역시 코딩 문제는 묻지만 연구 실적 등도 주의 깊게 살펴본다. PM은 영어가 힘든 나에게는 해당 사항이 아니었고, 따분해 보이는 SDET보다는 역시 제대로 된 개발을 할 수 있는 SDE로 지원을 하였다. 예를 들어, SDE 포지션이 요구하는 조건은 홈페이지에 따르면 다음과 같다.[4]

---

1. www.MonsterTRAK.com를 참고하라.
2. http://www.microsoft.com/college/YourSchool.aspx를 방문해보라.
3. SDET는 '에쓰뎃'이라 발음함. 영어가 항상 고민인 한국 학생들에게 이렇게 에쓰뎃이라고 말하면 처음에 알아듣기가 쉽지 않다.
4. http://www.microsoft.com/college/default.mspx에는 마이크로소프트 인턴쉽에 대한 좀 더 자세한 설명을 볼 수 있다.

- 전산학 및 관련 공학 분야에서 학사, 석사, 그리고 박사 학위 과정에 있어야 하며
- 1~2년의 C/C++/C#, 자바 및 기타 컴퓨터 프로그래밍 언어 경험 우대
- Thinking "Outside the box"[5]를 통해 창조적이고 혁신적인 해법들을 도출할 수 있는 능력
- 사물을 정의하고, 디자인하며, 실행 가능한지를 따지는 데 능숙해야 하며
- 개발 시간을 추측하는데 있어서 합리적인 기술

보다시피 윈도우 플랫폼에서의 개발 지식을 명시하지 않고 있다. 윈도우를 단 한 번도 사용하지 않은 사람조차도 MS에 취직하는 데 원칙적으로 전혀 문제가 없다. 그들이 생각하는 중요한 능력은 일반적인 전산 능력인 것이다. 그리고 MS가 운영하는 Job blog[6]에서도 평소 리눅스를 좋아하는 한 학생의 취업 이야기도 읽을 수 있었다.

서론이 꽤 길었다. 이제 실제 내가 두 번에 걸쳐 겪은 마이크로소프트 개발자 인턴 면접 과정을 이야기 해보자. 먼저 말하고 싶은 것은 MS는 인턴 면접자에게도 상당히 많은 돈과 시간을 투자한다는 사실이다. 즉, MS는 인턴을 뽑아도 정직원을 뽑는다는 생각으로 뽑는다. 그리고 실제로 정직원의 채용 과정도 인턴과 크게 다르지 않다.

1차 면접

내가 아는 한 MS/구글/아마존은 1차 면접을 보통 학교에서 직접 본다. 즉, 학교에 면접관이 찾아와서 면접을 본다. 그렇지 않으면 전화 면접으로 대체하기

5. Thinking outside the box는 기존의 틀에서 벗어나 좀 더 창의적인 관점에서 문제를 살펴보자는 표현이다. Wikipedia에서 좀 더 자세한 설명을 볼 수 있다.
6. http://blogs.msdn.com/jobsblog/를 방문해보라. MS 취업에 대한 많은 이야기(면접에 관한 팁은 물론)를 볼 수 있다.

도 한다. 혹은 IBM Extreme Blue 인턴쉽 프로그램처럼 화상 면접이나 컨퍼런스 콜로 면접을 보는 곳도 있다.

일단, 앞에서 이야기한 것처럼 이력서를 던져 주고 잊혀질 때쯤 되면 학교에서 일단 얼굴 한 번 보자고 연락이 온다. 나는 박사 과정의 진학을 염두에 두고 있었고, 또 1년차라서 그런지 적극적으로 이력서를 제출하지 않았다. 주로 소프트웨어 개발 쪽으로 3~4 군데 이력서를 내었지만, MS만이 먼저 연락을 주었다. VMware 같은 회사도 나중에 연락을 주었지만, 나의 경력과 맞는 부서가 당장 없어서 추후에 다시 지원하라는 이야기를 들었다. 이력서를 낸다고 해서 모두 1차 면접의 기회를 얻는 것이 아니다. 나 같은 경우는 아무래도 윈도우에서의 개발 경력이 많아서 어렵지 않게 기회를 잡았던 것 같다. 물론 전혀 윈도우 프로그래밍 개발 경험이 없는 친구들도 면접 기회를 많이 잡기 때문에 이런 점에 대해 염려할 필요는 전혀 없다. 앞서도 말했지만 Visual Studio를 한 번도 켜보지 않은 사람도 마이크로소프트에 갈 수 있다.

1차 면접은 흔히 screening test라고 불린다. MS의 경우 1차 면접은 약 30분 정도 할당이 된다. 간단한 자기 소개를 한 뒤, 코딩 및 퀴즈 문제를 풀고, 마지막에는 Q&A 시간을 가진다. 나와 면접을 본 면접관은 MS에서 15년 동안 오피스 등 여러 프로그램을 개발한 프로그래머였다. 깜짝 놀랐다. 직접 고참급 엔지니어가 올지는 몰랐다. 그는 양쪽 귀에 귀걸이가 무려 5개나 박혀 있었고 분홍색 남방을 입은 키 큰 유럽풍 남자 아저씨였다. 일단 편안하게 맞이 해준다.

그러고 보니 영어로 면접을 보는 것은 처음인 것이었다. 정말 영어 때문에 늘 고생인 우리 한국인 유학생들로서는 면접 보기 전에 걱정이 이만저만이 아니었다. 그나마 전화 면접이 아니라서 다행이었다. 리쿠르터와 같은 사람들의 전화상 발음은 괜찮지만(그들도 우리가 영어를 힘들어 한다는 것을 잘 안다) 일반적인 전화 통화(예를 들어, 보험 회사에 전화를 걸어 상담원과 이야기하는 것)는 정말 힘들기 때문이다.

1차 면접의 시작은 자기 소개로 시작된다. 특히 지난 학기에 윈도우상에서 보안 관련 프로젝트를 한 것을 중점적으로 설명했다. 면접관도 흥미를 보이면서

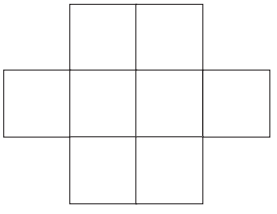
여기에 관한 여러 가지를 묻는다. 예를 들어, 내가 겪은 어려움을 이야기를 하면, 면접관이 "아 그래? 그러면 이런 문제도 있었을 것인데 어땠어?"라고 물어 보고 이야기를 술술 풀어나갈 수 있었다. 이렇게 이야기를 주고받으니 어느덧 15분이 훌쩍 지나갔다. 이렇게 내가 알고 있는 것, 한 것을 말하는 경우에는 영어도 크게 어렵지 않다. 이제는 한국의 면접 분위기도 더 이상 겸손 떠는 것은 아니니 자신 있게 자신이 한 일을 잘 광고해야 할 것이다. 단순히 이것저것 만들었다가 아닌 구체적으로 어떤 어려움이 있었고, 이 문제를 어떻게 극복했는지를 설명하는 것이 중요하다.

그러나 문제는 코딩 및 퀴즈였다. 종이와 연필을 던져주고 간단한 코딩 문제를 낸다. 쉬운 문제였지만, 그 자리에서 막상 보면 잘 생각이 나지를 않는다. 이 문제를 그냥 학교 숙제라고 생각하고 집에서 조용히 컴퓨터 앞에서 푼다면 누구나 다 풀 것이다. 사실 가장 힘든 것은 이것을 영어로 또박또박 설명하는 것이다. 우리말로 설명해도 쉽지 않은 데 말이다. 미리 충분한 연습이 필수다. 아는 것도 다시 한 번 정리할 필요가 있다. 그리고 바로 여러분이 손에 쥐고 있는 Programming Interviews Exposed는 매우 유명한 책으로 프로그래밍 인터뷰를 앞둔 학생들에겐 필독서로 되어있다.

다행히 문제는 어렵지 않게 풀었다(알고 보니 이 책에 있던 문제였다!). 알고리즘을 대략 말로 설명해도 되지만, 생각보다 꼼꼼하게 모든 조건을 언급하기를 요구한다. 물론 가장 완벽한 답안은 완벽한 코드를 쓰는 것이지만 면접관에 따라 그런 것을 항상 요구하는 것은 아니다. 그리고 다른 사람의 경우에는 Sudoku 같은 퍼즐을 풀게 하는 경우도 있다.[7]

---

7. 오른쪽 8개의 상자에 1~8까지 채워 넣는데, 인접한 사각형들에는 인접한 수가 들어갈 수 없다는 제약 조건을 지키면서 넣어보라.



또, 경우의 수를 묻는 경우도 있다. 3×5 격자로 된 바둑판의 왼쪽 아래에서 오른쪽 끝까지 도달할 수 있는 경우의 수는 몇 개나 있을까?

이 두 문제에 대한 정답은 맨 뒤에 있다.

그러나 맨홀 뚜껑은 왜 둥그런가와 같은 수수께끼 스타일은 더 이상 내지 않는다.

그 뒤에는 Q&A 시간이 주어지는데, 그냥 "질문 없어요"라고 대답하면 안 된다. 나는 원래 윈도우에 관심이 많은 터라 이런저런 질문을 많이 했다. 특히 독자적으로 쓰는 개발 툴이나 디버깅 툴이 있는지 물어봤는데, 그런 것은 별로 없다는 대답을 들었다. 나를 면접본 분은 커널이나 시스템 레벨이 아닌 애플리케이션 레벨이다 보니 특별한 툴은 없는 것 같았다. 그 외에 이런저런 이야기를 주고받으니 30분이 훌쩍 지나간다. 그리고 Microsoft가 찍힌 볼펜을 가지고 싶은 만큼 들고 가라면서 헤어진다. 답변은 2주 안으로 받게 된다고 알려주고, 1주 반 정도 지나면 결과가 나온다.

2차 면접

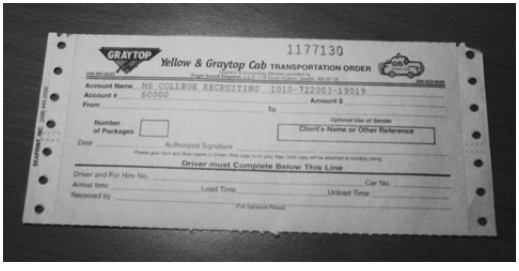
MS의 채용 과정 중 재밌는 것 하나가 있다. MS의 인턴 채용은 각 단계가 넘어가면 리쿠르터가 계속 바뀐다. 그러니까 1차 면접 일정을 잡아주던 리쿠르터와 2차 면접 일정을 조정하는 리쿠르터가 다르다. 또한, MS 본사가 있는 Redmond에서 만나는 리쿠르터 역시 다르다. 앞글에서 말했듯이 게으른 리쿠르터를 만나면 한 없이 기다리는 일도 벌어진다. 큰 규모의 회사에서는 채용 과정이 이렇게 길 수밖에 없다. 그러나 박사 과정의 연구 자리는 의외로 짧게 끝나는 경우도 많다. 그 때는 일반적인 리쿠르팅 과정이 아닌 연구소 특정 팀장이 직접 학생과 전화 면접으로 결정하기도 한다. 그래서 1~2주면 다 끝나기도 한다.

2차 면접은 직접 Redmond 메인 캠퍼스에서 면접을 보기 때문에 On-site interview라고도 불린다. 일단 마이크로소프트의 인턴 및 풀타임 인터뷰의 가장 큰 장점은 비행기/호텔 비용은 물론 관광 비용까지 전액 MS가 부담해준다는 사실이다. 아무리 작게 잡아도 천불이 넘는 돈을 면접자에게 쓴다는 소리다. 뒤에서도 자세히 쓰겠지만 수많은 리쿠르터들이 면접자들의 편의를 위해 노력

하고 있었다.

2차 면접은 직접 내가 일을 하게 될 팀의 팀원들과 직접 면접을 본다. 팀 배정은 순전히 리쿠르터 혹은 1차 면접관 마음대로인 것 같다. 나는 생긴지 6개월 된 팀과 면접을 보게 되었다. 이 팀은 C#으로 특정 소프트웨어 시스템을 만드는 것이었다. C#으로 이런 성격의 프로그램을 만든다는 것이 다소 의외였다. 그런데 나는 C#으로 개발한 적이 없었다. 아마도 내가 예전에 수행했던 프로젝트 중 비슷한 것이 있어서 이렇게 고른 것 같다. 재미있었던 것은 나는 그 어떤 사람에게서도 "C#을 써봤느냐"라는 질문을 받지 않았다. 오직 기본적이 전산 능력을 중점적으로 테스팅하였다.

원하는 시애틀 방문 날짜는 자기가 최대한 고를 수 있다. 그리고 공식적으로 2박 3일 동안의 경비를 지원한다. 첫째 날 도착해서, 둘째 날 면접보고, 마지막 날 관광하고 집으로 돌아가는 구조다. 그러나 하루 더 연장해서(3박 4일) 하루 종일 관광만 할 수도 있다. 최종 면접 날짜가 확정이 되면 MS 여행 부서에서 비행기 표와 호텔을 예약해서 알려준다. 교통수단은 렌터카나 택시를 고를 수 있는데, 렌터카를 선택하면 역시 공항에서 바로 차를 탈 수 있게끔 해준다. 인터뷰 내내 느꼈지만 정말 면접자를 최대한 편하게 해주려고 많은 리쿠르터들이 노력하고 있었다.



나는 렌터카 대신 택시를 선택하였다. 위 그림은 MS에서 주는 택시 바우처(Voucher)다. 일종의 쿠폰으로, Yellow Cab 택시를 부르고 이 바우처를 주면 택시비를 낼 필요가 없다. 물론 다른 택시를 잡아타더라도 돈은 다 돌려준다.

그 외 관광(박물관 관람 비용 포함)에 소요되는 모든 경비는 전부 다 되돌려 받을 수 있다. 식비도 하루에 75불까지 돌려받을 수 있다. 물론 도박이나 옷 등을 사는 데 쓰는 돈은 돌려받을 수가 없다고 나와 있다.

면접은 오전 11시부터 오후 4시까지 진행된다. 나는 한 팀하고만 면접을 보았고, 총 4명의 직원과 면접을 하였다. 두 팀과 면접을 보는 경우도 있으며, 많게는 5명과 면접을 보는 경우도 있다. 한 사람 당 한 시간씩 면접을 보는 구조다. 물론 중간에 점심 시간이 있으며, 두 번째 면접 본 친구와 같이 밥을 먹으면서 이런 저런 이야기를 한다. 점심을 먹는 과정도 면접의 일부이므로 긴장을 늦추면 안 된다.

먼저 리쿠르팅 빌딩인 19동으로 아침 10시까지 도착한다. 인턴 면접은 거의 끝 무렵이라 그런지 면접 보러 온 친구들은 많지 않은 것 같다. 이미 인턴을 시작한 친구들도 있으니 말이다. 여기서 굉장히 친절한 여자 직원이 과도한 친절을 보이며 내일의 관광 일정을 잡아준다. 이윽고 담당 리쿠르터가 나오면서 오늘의 면접 일정에 대해 알려준다. 면접은 실제 직원들과 만나는 것이므로 그들이 있는 빌딩으로 간다. MS 캠퍼스는 굉장히 넓으므로 모든 이동은 전용 차량으로 움직인다.

도요타의 전기-가솔린 하이브리드 차량으로 면접자를 각 건물로 이동시켜 준다. 면접자를 위해 이렇게 적지 않은 전용 차량까지 준비하고 있다는 것이 놀라웠다. 하나 재밌던 사실은 운전하시는 분들이 모두 나이 많으신 할아버지/할머니라는 사실이다. 어르신들에게 간단한 일자리를 주는 셈이다.

그리고 11시부터 본격적으로 면접을 본다. 1차 면접과 비슷하게 간단한 자기소개를 시작으로 코딩 문제가 핵심을 이루고, 마지막에는 Q&A가 뒤따른다. 코딩 문제는 직접 화이트보드에 코드를 써가며 설명을 해야 한다. 이런 스타일의 면접은 참 쉽지가 않다. 이에 반해 지식을 묻는 면접은 이보다 훨씬 수월하다. 예를 들어, "세마포어와 뮤텍스의 차이가 뭐야?"라고 물어보는 질문에 대답하는 것은 쉽다(모르면 큰일나지만). 또한 경험을 묻는 질문 역시 수월하다. 예를 들어, "네가 겪은 버그 중 가장 힘든 것이 뭐였어?" 같은 것도 힘들지 않다. 이 문제들은 이번에 내가 직접 받은 질문이기도 하다.

알고리즘 문제는 기출 문제 등으로 준비를 할 수는 있지만, 직접 그 자리에서 머리를 써야 하기 때문에 결코 쉽지 않다. 그렇다고 해서 아주 어려운 문제를 내는 것은 결코 아니다. 풀지 못하더라도 문제에 어떻게 접근하는지 사고 과정을 잘 보여주는 것이 중요하다. 역시 영어로 내가 만든 알고리즘을 정확하게 표현하는 능력이 가장 힘들었다. 그리고 각 면접이 끝나면 면접관이 각종 정보를 다음 사람에게 모두 넘겨준다. 그래서 앞 면접에서 내가 어떤 짓(?)을 했는지 모두 알고 있다.

처음 만난 친구는 이제 들어 온 지 얼마 되지 않은 미국 친구였다. 말을 또박

또박 해줘서 알아듣는 데 큰 문제는 없었다. 코딩 문제도 친절하게 종이에 프린트까지 해서 나에게 주었다. 처음엔 긴장해서 약간 잘못 이해하기도 했지만, 큰 어려움 없이 풀 수 있었다. 그리고 미처 생각하지 못한 부분을 계속 집어내며 나의 답변을 얻었다. 이 친구에게는 특이하게도 소프트웨어 디자인 문제도 받았다. 단순히 알고리즘 문제가 아니라 실제 이 팀에서 만드는 제품 중에서 이 기능을 넣을 때 어떤 방식으로 할 것인가와 같은 문제도 받았다.

두 번째 만난 분은 마치 영화 주라기 공원에 나왔던 뚱보 프로그래머와 같은 분이었다. 방이 굉장히 지저분하였고, 휴지통에는 콜라 캔이 산더미처럼 쌓여 있었다. 이 분과 풀은 문제는 그렇게 쉽지 않았다. 생각보다 이 분은 아주 꼼꼼한 코드까지 요구해서(예를 들어, 인덱스 변수가 i+1인지 i인지와 같은 것) 꽤나 땀을 흘렸다. 이렇게 두 번째 면접을 마치니 어느덧 오후 한 시였다. 이 친구와 함께 밥을 먹으러 갔다.

밥을 먹으면서는 비교적 가벼운 대화를 나누었다. 나는 아쉽게도 어학연수를 갖다 온 적도 없고, 외국인 친구도 많은 편이 아니어서 이런 가벼운 대화가 상당히 힘든 편이다. 차라리 무엇을 설명하라는 것은 자신 있지만 이런 대화는 쉽지 않다. 그래도 밥을 먹으면서 많은 대화를 나누었다. 특히 어떻게 소스 코드를 관리하는지 내가 물어봤는데, 각자 가진 스마트 카드키로 소스 코드에 접근이 가능하다고 한다. 그리고 이것만 있으면 집에서도 근무가 가능하다고 한다. 그래서 어떤 날은 회의가 없는 날로 정해 재택근무도 종종 한다고 들었다. 그리고 이 팀이 C#으로 소프트웨어를 개발하지만 특별히 더 멋지고 편한 라이브러리를 쓰는 것은 아니라고 한다. 그냥 다른 회사 사람들이 C#을 가지고 개발하듯이 자기들도 그렇게 할 뿐이라고 이야기했다. 단지 차이점이 있다면, .NET 프레임워크의 소스를 볼 수 있는 권한이 있다는 것이었다.

세 번째로 만난 친구는 중국계 친구였다. 택시를 타면서 택시 아저씨가 MS에서 인도와 중국 사람들이 빠지면 바로 망한다는 소리를 할 정도로 MS에는 외국인이 많다. 특히, 중국과 인도인들이 많았다. 그리고 MS는 외국인을 가장 많이 고용하는 회사 중 하나다. 2007년 현재, 미국에서는 취업을 위한 비자인

H1B 쿼터를 두고 말이 많다. 빌 게이츠 회장은 직접 국회 청문회에 나와 더 우수한 외국 인재를 잡을 수 있도록 H1B 숫자를 늘려야 한다고 주장하였다. 빌 게이츠의 생각대로 MS는 외국인을 위한 H1B, 그리고 영주권 신청에도 상당히 적극적이다(물론 보수적인 공화당 의원들로부터는 비난을 받는다).

마지막에 만난 분은 팀장이었다. 12년 동안 여기서 일했다고 하는 인도 출신의 개발자였다. 굉장히 조용히 말하였고, 특히 말을 알아듣기 너무 힘들어서 고생을 많이 하였다. 레쥬메를 자세히 읽어본 것 같았다. "예전에 네트워크 프로젝트를 했던데, 그거 설명해봐"라는 식으로 이야기를 시작한다. 그리고 "리눅스와 윈도우의 차이가 뭐라고 생각하느냐"라는 질문도 들었다. 그래서 평소 가지고 있던 이야기를 잘 풀어 이야기를 했다.

이 아저씨와는 질문 답변을 꽤 많이 주고받았다. 특히 외국인으로서 여기에서 사는 것이 어떠냐고 물어봤더니, 인도에서의 삶과의 trade-off는 있지만 행복하다고 말씀하셨다. 그런데 얼굴 표정이나 말투를 봐서는 전혀 행복해 보이지 않을 정도로 딱딱한 인상이었다. 내가 그 분에게 한국에서는 10년 이상 된 고참 프로그래머(seasoned programmer)가 없고 대부분 매니지먼트로 빠지는 현실을 말했더니, 인도도 그렇다고 말씀하신다. 반면 MS에는 10년 넘은 개발자를 많이 볼 수 있다. 그리고 그들이 반드시 PM으로 가야 한다는 생각도 하지 않고 있었다. 왜냐면 MS는 PM과 개발자가 수직적인 관계가 아닌 수평 구조이기 때문이라 말씀하셨다.

위에서도 말했듯이 이 팀은 만들어진 지 이제 5~6개월이 된 팀이었다. MS는 팀 간 이동도 비교적 수월한 편이라고 한다. 다른 팀에서 받아주는 과정은 내가 겪은 면접 과정을 동일하게 거쳐야 한다고 한다. 그래서 이직을 하고 싶으면 팀을 옮기면 되기 때문에 MS에 계속 머물게 된다고 말씀하셨다.

4명의 개발자와의 인터뷰를 끝으로 면접은 끝나게 된다. 그리고 별도로 저녁 시간에는 MS의 다른 개발자와 저녁 식사 자리도 마련해준다. 나 같은 경우에는 나와 같은 학교를 졸업한 분이 배정이 되었다고 한다. 그러나 너무 피곤한 관계로 취소하였다.

면접을 마치고 나서

Redmond 메인 캠퍼스에는 3~4만 명이나 근무하기 때문에 상당히 큰 규모다. 그다지 여유가 없어서 캠퍼스를 자세히 구경하지는 않았는데, 캠퍼스 자체가 아주 아름답거나 그러지는 않다. 그냥 단정한 4~5층 규모의 건물들이 많이 들어서 있었다. 그리고 곳곳에는 공사가 벌어지고 있었으며, 캠퍼스는 계속 확장하고 있는 중이었다.

많이들 알고 있겠지만 MS 개발자들은 각자의 방을 혼자 사용한다. 물론 한 방을 두 명이 사용하는 경우도 있지만, 계속 건물을 짓고 있으니 1인당 1방이 기본인 것 같았다. 첫 번째 만난 친구는 무려 모니터를 3개나 쓰고 있었다. 방 크기는 결코 좁다고 느낄만한 수준은 아니었다. 보통 모니터 두 대 정도는 가지고 있었다.

구글은 레스토랑을 방불케 하는 화려한 식당으로 유명하다. 게다가 음식 값이 모두 무료이다. 방문자 역시 무료이다. 사진을 찍지는 못했지만 MS 식당도 생각보다 매우 다양한 메뉴와 적어도 중급 이상의 레스토랑 수준은 되는 음식을 제공하고 있었다. 나는 "커리 치킨 그릴"을 시켰는데 그 자리에서 생고기를 직접 구웠다. 온도계를 고기 속 내부로 찔러 온도까지 확인하였다. 이런 건 사실 첨 봤다. MS 직원들은 스마트카드 비슷한 걸로 결제를 하고, 면접자는 쿠폰으로 돈을 대신한다. 대략 4불정도 하는 것 같았다. 학교 식당이 보통 5불을 쉽게 넘어가는 걸 생각하면 저렴하다고 볼 수 있겠다.

그다지 놀라운 것은 아니겠지만, MS 빌딩에는 각 층마다 냉장고에 음료수가 무료로 제공된다. 몇몇 블로그에서 볼 수 있었던 윈도우 비스타 소다류가 있나 찾아보았는데 없었다. 대신 간단한 초콜릿이나 스니커스바 같은 것은 자판기로 구입하도록 되어있었다. 얼마인지는 확인하지 못하였다. 그리고 에스프레소 기계도 보였고, 1층에는 미국 드라마 같은 곳에 보면 종종 볼 수 있는 게임대 같은 것도 보였다.

캘리포니아 실리콘 밸리 산호세 옆에 있는 Mountain View라는 도시에 있는 구글은 통근 버스가 아주 잘 되어있기로 유명하다. 거의 시내버스를 방불케 할 만큼 잘 되어있다. 시애틀과 그 인근 도시인 Redmond는 대중교통이 그리 잘 되어있는 편이 아니다. 지하철 따위는 없다. 뉴욕을 제외하고, 미국에서 서울과 같은 거미줄 같고 게다가 저렴하기까지 한 지하철과 버스를 생각하면 절대 안 된다. 웬만한 우리나라 광역시보다도 훨씬 못하다고 보면 된다. 원래 미국이라는 나라는 자동차가 없으면 거의 생활이 불가능한 경우가 많기 때문이기도 하다. 그래서 역시 MS도 자체 통근 버스를 운영한다고 택시 아저씨가 말하는데, 잘 되어있다고 칭찬을 하였다. 아쉽게도 직접 본적이 없어서 구체적으로 어떤지는 알 길이 없었다.

보다시피 MS는 인턴 하나 뽑는 데도 많은 돈과 시간을 투자함을 알 수 있다. 앞에서도 말했듯이 인턴을 뽑는 것이나 정직원을 뽑는 것이나 큰 차이가 거의 없다. 그래서 인턴으로 뽑힌 뒤, 일을 잘 하면 바로 풀타임 자리를 받을 수 있고, 이것은 흔히 있는 일이다. 세 번째 만난 친구도 4년 전에 인턴으로 온 뒤에 여기 계속 남아있는 것이라고 말했다. 그러니까 MS에 취직하기 가장 쉬운 방법은 인턴으로 가서 열심히 일하는 것이다.

MS에는 개발자만 3~4만 명이 있는 거대한 기업이다. 당연하겠지만 거기에 있는 모든 개발자가 전부 수퍼 guru급의 프로그래머인 것은 아니다. 모두 똑똑하고 그런 건 절대 아니었다. 비록 한국의 대학에서 바로 취직하는 경우는 여러 외적인 문제로 힘들겠지만, 미국에 있는 대학만 다닌다면 아주 어렵지는 않다. 물론 MS는 여러 해외 대학에서 리쿠르팅을 한다. Job Blog를 읽어보니

이집트에서 온 한 여학생의 글도 볼 수 있었다.

유학을 나오기 전 여러 한국 소프트웨어 업체와도 면접을 보았다. 역시 면접의 농도와 깊이는 정말 MS/구글과 같은 미국 유명 소프트웨어 회사와는 비교할 바가 되지 못한다. 4명의 직원과 1:1로 4시간 면접을 보았던 N모 게임소프트웨어 회사가 가장 근접하다고 볼 수 있겠다. 그러나 단순히 개발 지식만을 물었지 일반적인 알고리즘 테스트는 하지 않았다. 우리나라에서 가장 큰 N모 인터넷 기업도 시험을 보기는 하지만 심층적인 기술 면접은 경험하지 못하였다. 4명의 면접관과 3명의 지원자가 한 시간 동안 산만하게 질문을 주고받는 수준이다. A 보안 회사 같은 경우도 코딩 문제를 메일로 받아 풀어서 제출은 하기는 했지만, 직접 알고리즘을 화이트보드에 써가며 이야기하지는 않는다. 부사장 및 사장님과 함께 보는 2차 면접은 약간 부담스럽기는 하였지만 "얼마나 아느냐"보다는 얼마나 우리 회사에 잘 맞느냐를 보는 것 같았다.

내가 느끼기에 우리나라는 단순히 "개발 지식"에 초점을 맞춘다고 볼 수 있다. 반면 MS나 구글 같은 곳은 잠재적인 "개발 능력"을 더 따지는 것 같았다. 물론 윈도우 커널이나 검색 엔진의 개발과 같은 아주 특화된 팀에는 개발 지식도 상당히 중요할 것이다. 그러나 적어도 인턴을 뽑는 포지션에서 전문적인 개발 지식은 중요하지 않았다. 나 역시 C#을 써보지도 않았는데, 그 누구도 "너 C# 할 줄 아냐"라고 묻지 않았다. 우리나라 같으면 아무리 전산 지식이 풍부해도 당장 .NET 프레임워크를 다뤄보지 않았더라면 쉽게 뽑았을지 의문이 든다.

정리를 하면 2차 면접은 상당히 피곤하다. 세 번째 면접부터 이제 머리가 잘 돌아가지 않는다. 목도 쉬어서 말도 잘 안 나온다. 충분한 숙면과 중간중간마다 초콜렛바로 에너지를 채울 것을 권장한다. 역시 가장 중요한 것은 탄탄한 기본 전산 실력이다. 그리고 주어진 알고리즘 문제를 화이트보드에 써가며 논리적으로 설명해가는 능력이 중요하다. 그리고 한국 학생에게는 이것을 영어로 조리 있게 또박또박 천천히 표현하는 능력이 사실 가장 중요하다고 볼 수 있겠다.

실제 프로그래밍 인터뷰 문제들

이제 나와 나의 친구들이 MS 인터뷰에서 받은 문제를 정리해보자. 그 중 몇 문제에 대해서는 내가 생각하는 답안을 정리해보았다. 하나 일러두고 싶은 것은 질문의 난이도는 면접을 보는 팀에 따라 꽤 차이가 난다는 것이다. 일반적으로 알고리즘 문제를 묻는 것은 큰 차이가 없지만, 팀의 성격에 따라 난이도가 다를 수 있다. 인턴 포지션 중 테스팅을 전문으로 하는 SDET이라는 포지션은 자기가 짠 코드를 어떻게 디버깅 및 테스트할 것인가를 더 묻기도 한다.

참고로 구글과 같은 기업도 난이도의 차이가 있을 뿐 큰 흐름은 같다고 보면 될 것이다. 하나 알아두어야 할 것은 모든 소프트웨어 업체들이 이런 식으로 면접을 보는 것은 아니라는 점이다. IBM의 어떤 부서 같은 경우에는 마이크로소프트와 구글의 알고리즘 면접 방식을 그다지 좋게 생각하지 않는 곳도 있기 때문이다. 이 책의 14장에서 다루고 있는 지식 기반의 문제를 던지는 곳도 많다는 것을 알아두길 바란다.

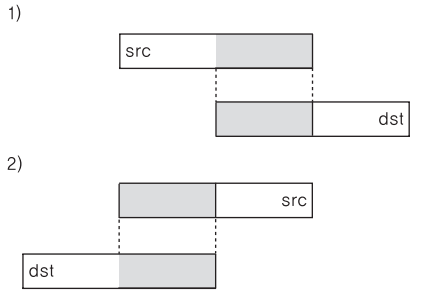
인터뷰 문제

C/C++로 프로그래밍을 해본 사람이라면 strcpy, memcpy, memmove와 같은 복사 함수를 사용해보았을 것이다. 그런데 예를 들어, 다음과 같은 memcpy 함수에서 dest와 src의 메모리 공간이 서로 겹친다면 어떻게 될까?

```
void *memcpy(void* dest, const void* src, size_t count );
```

실제로 C언어 표준 규약인 C99에서는 memcpy의 경우 두 메모리 공간이 겹쳐지는 것에 대해서는 정의를 하지 않고 있다. 즉, 어떤 일이 벌어질지 모르므로 항상 겹치지 않는 공간을 줘야만 한다(그래서 C99에는 "restrict"라는 qualifier가 양쪽 모두 붙어있다). 반면 memmove는 이런 겹치는 부분이 있을 때 알아서 잘 처리한다. 물론 이것을 처리하기 위한 비용은 더 들어갈 것이다. 그렇다면 memcpy도 겹치는 부분이 있다면 어떻게 처리하면 좋을까?

겹쳐지는 부분은 아래와 같이 두 가지 경우로 생각할 수 있을 것이다.



1)의 경우에는 src의 뒷부분과 dst의 앞부분이 겹친다. 이 경우에 src의 뒷부분은 결국 파괴될 것이다. 이 경우에는 뒷부분(높은 주소)부터 복사를 시작하면 될 것이다. 2)의 경우에는 반대로 낮은 주소부터 복사를 하면 될 것이다. 찬찬히 생각하면 어려운 문제는 아닐 것이다. 참고로 memmove의 소스는 아래와 같다.[8]

```
void * memmove(void * dst, void * src, size_t count)
{
   void * ret = dst;

   if (dst <= src || dst >= (src + count)) {
      /*
       * Non-Overlapping Buffers
       * copy from lower addresses to higher addresses
       */
      while (count--)
         *dst++ = *src++;
   }
   else {
      /*
       * Overlapping Buffers
       * copy from higher addresses to lower addresses
```

8. 이 소스는 Microsoft Visual Studio 2005가 설치하는 각종 파일 중 하나인 memcpy.asm 파일에서 발췌한 것이다.

```
         */
        dst += count - 1;
        src += count - 1;

        while (count--)
           *dst-- = *src--;
     }

     return(ret);
  }
```

그리고 여기에서 부가적으로 생각해볼 수 있는 문제는 작성한 코드가 Endianness(Little-endian/Big-endian)에 영향이 있는지, Alignment에는 영향이 있는지 정도가 있을 것이다.

인터뷰 문제

이 문제는 SDET로 면접을 본 친구에게 들은 문제이다.
"메모리 할당 함수인 malloc과 free를 만들었다고 하자. 어떻게 테스트할 것인가?"

일단 기본적인 메모리 할당이 제대로 되는지 살펴봐야 할 것이다. 할당한 메모리 블록들이 서로 겹치는 것이 없고 해제 시 문제없이 해제되는지 여러 가지 가능한 경우를 가지고 테스트를 해야 할 것이다. 여기까지는 누구나 생각할 수 있는 답변이고, 또 어떤 것을 테스트하면 좋을까?

먼저 성능을 따져봐야 할 것이다. 실제로 어떤 프로그램들은 malloc/free를 1초에 수백만 번 호출하는 경우도 있다. 이런 경우 이 함수들에서 병목 현상이 일어나는지 살펴봐야 할 것이다. 단순히 호출하는데 걸리는 시간뿐만 아니라, 이제는 듀얼코어가 기본인 환경이므로 멀티프로세서일 때의 성능도 언급하면 좋을 것이다. 실제로 일반적인 malloc/free의 구현은 멀티프로세서 환경에서 그다지 좋지가 않다.

특히 false sharing[9]이라는 문제가 발생하여 많은 프로세서가 있는 환경에서는 speedup이 좋지 않게 나온다. 따라서 이러한 것도 고려하면 어떨까라는 답변을 하면 멋질 것이다.[10]

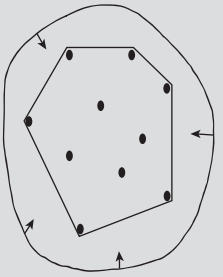
마지막으로 윈도우뿐만 아니라 리눅스 운영체제를 써도 보안 패치가 자주 있다. 그런데 이런 보안 패치의 대부분은 소위 말하는 heap overflow attack을 막기 위한 것이다. 할당 받은 크기보다 더 많은 데이터를 강제로 써서 프로그램의 제어권을 탈취하여 악성 코드를 실행케 하는 이 방법은 대표적인 소프트웨어 취약점 중 하나이다. 실제로 2001년을 떠들썩하게 했던 Cord Red Worm도 이것을 악용한 것이다. 일반적인 malloc/free는 사용자 데이터가 저장되는 공간의 사이사이마다 할당한 메모리 블록을 관리하기 위한 metadata가 존재한다. 그런데 이 metadata를 조작하면 악성 코드를 실행시킬 수 있는 희한한 일이 벌어진다. 그래서 이것을 막기 위한 많은 방법이 연구되고 있다. 그래서 메모리 할당 함수 군을 만들었다면 이 함수들이 이런 보안 취약성을 가지는 것에 대한 조사도 이제 필수일 것이다.

9. False sharing을 좁은 공간에서 설명하기는 쉽지 않다. 그래도 최대한 간략하게 설명하면, 보통 캐쉬 라인의 크기는 64바이트와 같이 여러 개의 데이터가 들어갈 수 있다. 그리고 여러 프로세서가 이 캐쉬 라인을 공유하면 이 단위로 공유가 된다. 그래서 같은 캐쉬 라인에 4바이트짜리 데이터가 두 개 있고 이 데이터가 각각 다른 CPU로부터 사용이 된다면, 불필요하게 이 캐쉬 라인을 invalidate시키는 문제가 발생한다(이 부분을 제대로 이해하려면 cache coherence protocol 중 대표적인 MESI 프로토콜을 알아야 한다). 생각보다 이 문제는 성능에 심각한 영향을 미친다.

10. 멀티프로세서 환경을 고려한 대표적인 메모리 관리자로 Hoard memory allocator가 있다. http://www.hoard.org/를 방문해보라.

인터뷰 문제

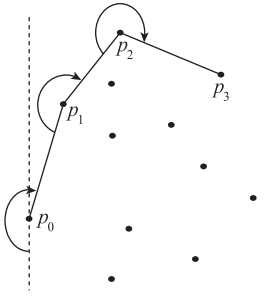
이번에는 convex hull에 관한 문제이다. 그래픽스 수업을 듣지 않은 사람이라면 convex hull 자체에 대해서도 생소할 것이다. convex hull은 수학적인 의미도 있지만 좀 더 직관적으로 그림으로 설명하면 주어진 점들을 감싸는 최소한의 공간을 뜻한다. 2차원 같은 경우는 아래 그림과 같다.[11]



그림에서 보다시피 오목한 부분 (concave)가 없고 볼록한 부분 (convex)만 있다. 그렇다면 이 convex hull을 어떻게 하면 구할 수 있을까?

이 문제는 이미 오랫동안 연구된 것이고 좋은 time complexity를 가지는 알고리즘도 많이 발견되었다. 그러나 실제 면접 볼 때 이런 천재적인 알고리즘을 요구하는 것은 아니다. 가장 간단한 알고리즘부터 차근차근 만들고 좀 더 개선할 여지가 있는지를 생각해서 말하면 좋을 것이다. 여기서는 가장 간단한 방법만 하나 소개할 예정이다. 그리고 좀 더 자세한 내용은 흔히 CLRS 책으로 불리는 유명한 알고리즘 책인 『Introduction to Algorithms』(By Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford, MIT Press)를 참고하기 바란다.

가장 간단한 방법 중 하나로 "Jarvis's march(혹은 Gift wrapping)" 알고리즘이 있다. 말 그대로 감싸듯이 convex hull을 찾는 알고리즘이다. 아래 그림은 이 알고리즘이 어떻게 작동하는지 간단하게 보여주고 있다.[12]



11. http://en.wikipedia.org/wiki/Convex_hull
12. http://en.wikipedia.org/wiki/Gift_wrapping_algorithm

먼저 가장 왼쪽에 있는 점인 P_0를 찾는다. 그리고 다음 점 P_1은 P_0와 이루는 각이 가장 작은 녀석을 고르면 된다. 이 과정을 반복하면 convex hull을 찾을 수 있다. 앞서 소개한 CLRS 책에는 이와 다른 방법으로 알고리즘을 설명하고 있다. 다소 다르기는 하지만 전체적인 맥락은 차이가 없다. 다만 지금 그림에서 소개한 방법은 마지막으로 선택된 점의 각을 기억해야 한다. 그리고 하나 생각해볼 것은 가장 작은 각도를 만드는 점이 하나가 아니라 여러 개일 때 이것을 어떻게 해결할 것인가도 생각해보면 좋을 것이다.[13]

이 알고리즘의 시간 복잡도를 계산해보자. 전체 점들의 개수를 n개라고 할 때, 다음 점을 찾는 과정은 각도를 구한 뒤 가장 작은 녀석을 찾으면 되므로 O(n)만큼의 시간이 걸릴 것이다. 그리고 이 과정을 convex hull에 포함될 점들의 개수 h만큼 수행하게 될 것이다. 따라서 알고리즘의 수행 시간은 O(nh) 꼴이 된다. 그래서 최악의 경우에는 h가 n과 같을 수가 있으므로 그 때는 $O(n^2)$ 알고리즘을 얻게 된다. 반면 h가 logn과 비슷하다면 이 방법은 상당히 우수한 성능을 보일 것이다. 그러나 알고리즘이 이렇게 미리 예측할 수 없는 h에 의존적이므로 항상 좋은 성능을 주는 것은 아니다. 반면 Graham's Scan 방식은 O(nlogn)이라는 시간을 준다.

인터뷰 문제

주어진 두 정수를 나누는 함수를 만드는 문제이다. 물론 나눗셈 연산자(/)는 사용할 수 없다.

```
int my_divisor(int a, int b);
```

알고리즘은 간단하게 루프를 돌면서 a에서 b를 계속 빼면 몫을 구할 수 있다. 여기까지는 간단하나 여러 예외 사항을 잘 고려해야 한다. 일단 0으로 나눠지는 경우는 손쉽게 검사를 할 수 있다. 또한 주어진 수가 음수라면 절대값으로 만든 뒤에 계산을 하는 것이 편할 것이다. 그런데 단순히 절대값을 만들기 위해 −1을 곱하는 것은 문제가 발생할 수 있다. 보통 정수 표현은 2의 보수법(2's

13. 이런 경우 영어로는 ~ breaks tie by selecting the furthest point ~와 같이 표현하면 된다.

complement)으로 표현을 하는데, 음수의 범위가 양수보다 하나 더 크다. 예를 들어, 8비트라면 −128부터 127까지 표현이 가능하다. 따라서 −128에 단순히 −1을 곱하면 잘못된 값을 얻을 수 있다. 이 부분 역시 적절히 고려를 해야 할 것이다.

인터뷰 문제

자연수 배열과 어떤 자연수 하나가 주어졌다. 이 배열에서 두 수의 합이 주어진 수와 같은 쌍이 있는지를 판별하라.

문제는 간단하다. 예를 들어, {1, 7, 4, 9}라는 배열과 13이라는 숫자가 주어지면 4+9가 13을 만족하므로 답은 true가 될 것이다. 반면에 20이라는 숫자가 주어지면 만족하는 쌍이 없으므로 false를 반환하다. 혹은 하나 이상의 조합이 나올 수도 있는데, 이런 경우에는 어떤 것을 먼저 보여줘도 무방하다(이런 이야기를 해주지 않았으면 자기가 찾아서 이런 경우는 어떻게 하면 좋은지 직접 이야기를 하는 것이 좋다. 항상 면접 보는 사람과 많은 이야기를 하는 것이 중요하다).

가장 간단하게는 $O(n^2)$ 알고리즘으로 모든 쌍에 대해서 검사를 하는 것이다. 물론 이런 알고리즘을 요구하지 않는다. 이제 좀 더 좋은 성능을 위해 O(n) 만큼의 공간을 이용하는 방법을 생각해보자. 만약 hash table 자료 구조를 이용할 수 있다면 O(n) 시간에 답을 쉽게 구할 수 있다. hash table에 자료를 넣거나 검색하는 것은 상수 시간에 완료되기 때문이다. 그렇게 어렵지 않으므로 여기에 코드는 적지 않는다.

그러면 공간을 더 활용하지 않고 좀 더 똑똑히 할 수 있는 방법은 없을까? 만약 주어진 배열이 오름차순으로 정렬이 되어있다면? 이런 경우 공간은 더 사용하지 않으면서 O(n) 시간에 답을 구할 수 있는 방법이 있다.

먼저, 가장 작은 원소인 첫 번째 원소를 가리키는 왼쪽 인덱스 변수와 가장 큰 원소인 마지막 원소를 가리키는 오른쪽 인덱스 변수를 두자. 그리고 이 둘의

합을 구한 뒤 주어진 수와 크기를 비교하자. 만약 합이 주어진 수보다 작다면 왼쪽 인덱스를 증가시키고, 크다면 오른쪽 인덱스를 감소시킨다. 그리고 계속 반복을 하면서 같아지는지 확인을 하자. 루프는 최대 배열 원소의 개수만큼 반복하기 때문에 O(n) 시간에 완료된다. 코드는 아래와 같다.

```
bool FindSumOfPair(int A[], int N, int K)
{
    int l = 0;
    int r = N - 1;
    int sum;

    while (l < r)
    {
        sum = A[l] + A[r];
        if (sum < K)
            ++l;
        else if (sum > K)
            --r;
        else
            return true;
    }

    return false;
}
```

만약 정렬이 comparison에 기초한 방법(Quicksort, Mergesort 등)이라면 최소 O(nlogn) 만큼의 시간이 걸릴 것이다. 결국 정렬 알고리즘이 시간 복잡도를 dominate하기 때문에 이 문제는 최고 O(nlogn) 시간에 해결할 수 있을 것이다. 반면 bucket sort처럼 O(n) 시간에 정렬을 할 수 있는 조건이라면 이 문제는 O(n)에도 풀 수 있다.

참고로 이 문제를 푸는 데 도움을 주신 iwongu, 시즈하, 에이쥬어, 그리고 silverbird님께 감사의 말씀을 전한다. 지금 생각하면 어렵지 않은 문제이다.

아니 쉬운 문제이다. 그런데 3시간 동안의 면접으로 지친 가운데 이런 문제를 또 받으면 머리가 아파서 더 생각하기가 귀찮을 정도였다. 사실 나는 hash table까지 이용하는 답이면 충분하다고 생각했는데, 난데 없이 추가 공간 없이 O(n)에 작동하는 걸 생각해보라는 질문을 듣고 당황하였다. 역시 면접 보기 전날 충분한 숙면과 컨디션 유지는 필요 조건임을 다시 깨달았다.

인터뷰 문제

n개의 비트로 이루어진 2진수에서 연속된 1이 나타나지 않는 경우는 몇 개나 있을까?

이 문제는 코딩 문제라기 보다는 퀴즈에 가까운 문제라고 볼 수 있다. 그러나 이 문제는 결코 간단하지 않는 귀납적 사고 능력을 요구하고 있다. 이런 귀납적 사고 능력은 실제 프로그래밍에서도(예를 들어, dynamic programming) 아주 유용하게 사용할 수 있는 것이므로 이러한 능력을 기르는 것은 매우 중요하다. 보통 이런 내용은 전산학과 과목 중 이산 수학에서 다룬다. 그리고 12장의 내용과도 어느 정도 방향을 같이 하는 것이라고 보면 될 것이다. 실제로 이런 문제를 인터뷰에서 묻는 경우도 있었다.

예를 가지고 문제를 다시 설명해보자. 2비트라면 총 가능한 경우의 수는 00, 01, 10, 11로서 4가지이다. 여기서 1이 연속으로 나타나지 않는 경우는 00, 01, 10 총 3가지이다. 이렇게 n개의 0과 1로 이루어진 수열에서 연속된 1이 없는 경우의 수(이것을 C(n)으로 표현하자)를 찾는 것이 이 문제가 요구하는 답이다. 이런 문제의 시작은 일단 손으로 낙서를 해가며 규칙을 찾아야 한다. C(3)까지 한 번 직접 구해보자.

C(1) = {0, 1}: 2개
C(2) = {00, 01, 10}: 3개
C(3) = {000, 001, 010, 100, 101}: 5개

수학과 조금 친한 사람이라면 대략 여기서 바로 피보나치 수열임을 짐작하고 C(4)을 손으로 구해 확인한 후 정답으로 바로 말할 수 있을 것이다. 그러나 이

런 직관적인 방법 말고 좀 더 수학적인 방법으로 답을 유도해보자.

이런 순열의 경우 일반항을 구하기 위해서는 점화식을 찾는 것이 첫번째로 할 일이다. 점화식은 영어로 recurrence로 표현이 되며, 각종 재귀 알고리즘의 시간 복잡도를 기술하는 데도 많이 사용된다. 예를 들어, merge sort의 경우 T(n) = 2T(n/2) + O(n)으로 표현되고 Master's theorem을 통해 O(nlogn)을 얻을 수 있다(자세한 것은 이산수학 책과 알고리즘 책을 참고하기 바란다).

점화식을 찾기 위해 수학 시간에 배운 수학적 귀납법을 이용해보자. C(n)의 해를 가지고 있다고 가정하자. 그리고 이제 C(n+1)을 기존의 해인 C(n)로 표현해보자.

숫자가 하나 더 늘었다는 소리는 n개의 순열 앞에 0 또는 1을 더 붙이는 것과 같다. 먼저 0이 추가되는 경우를 생각해보자. 생각하기 쉽게 이미 구한 C(3)을 가지고 생각하자. 이미 구한 5개의 답에 0을 앞에다 붙여도 여전히 이들은 주어진 조건을 만족하며 답으로 남는다. 즉, C(3)의 해를 고스란히 가져올 수 있다. C(4)에는 {0000, 0001, 0010, 0100, 0101}이 해로 포함이 될 것이다.

이제 1이 추가되는 경우를 생각해보자. 마찬가지로 C(3)의 해에다가 1을 더해보자. 앞의 3가지인 000, 001, 010은 문제 없다. 그러나 마지막 두 개는 1이 앞에 올 경우 1100처럼 되기 때문에 조건에 위배된다. 여기서 약간의 눈썰미를 동원해보자. 1을 더해도 조건을 만족하는 수들은 1000, 1001, 1010의 3가지이다. 그런데 이들은 처음 시작이 10으로 시작함을 알 수 있다. 그리고 그 뒤에 붙는 00, 01, 10은 C(2)의 해와 정확히 일치함을 발견할 수 있다. 즉, C(2)의 해에다가 10을 덧붙이는 것도 C(4)의 해가 된다.

따라서 C(4)는 결국 C(3)에다 0을 덧붙인 경우와 C(2)와 01을 덧붙인 경우로 생각할 수 있다. 즉, C(n)은 C(n−1) + C(n−2)가 되는 것이다. 그리고 이것은 바로 피보나치 수열과 정확하게 일치한다. 따라서 최종 정답은 아래와 같다.

C(n) = C(n−1) + C(n−2) (단, n은 n〉2인 정수, C(1) = 2, C(2) = 3)

이 문제를 다시 살펴보면, C(n)을 풀기 위해 이미 풀어놓은 해들을 잘 활용했음을 볼 수 있다. 이것은 동적 프로그래밍의 접근 방식과 동일하다. 동적 프로그래밍 방법론의 일반적인 접근 방식은 다음과 같다.

먼저 문제 중 겹치는 부분이 있는가를 찾아야 한다. 풀이에서도 이미 보았듯이 C(n+1)의 해에는 C(n)의 해가 겹쳐있다. 이 관계를 잘 분석해서 점화식으로 표현을 한다. 그리고 bottom-up으로 값들을 차례대로 계산하면 원하는 답을 얻을 수 있다. 대표적으로 matrix-chain multiplication, longest common subsequence와 같은 문제가 있다. 단순히 리스트, 트리와 같은 자료 구조에 관련된 공부뿐만 아니라 동적 프로그래밍 혹은 greedy algorithm과 같이 알고리즘 방법론도 숙지해야 할 것이다. 앞의 convex hull 문제에서 소개한 CLRS 책에는 이런 방법론들이 아주 자세히 설명되어 있다.

자, 그러면 응용 문제를 하나 더 풀어보자. 주어진 문제는 "연속된 1이 나타나지 않는 경우의 수"였다. 그러면 이 반대인 "연속된 1이 나타나는 경우의 수"는 어떻게 될까? D(n)을 구하고자 하는 수라고 표현하면 다음과 같을 것이다:

D(1) = {ø}: 0개
D(2) = {11}: 1개
D(3) = {011, 110, 111}: 3개
D(4) = {0111, 0110, 0111, 1011, 1100, 1101, 1110, 1111}: 8개

여기서 한 번 D(n)의 점화식을 구해보자. C(n)을 구하는 것보다는 약간 어렵다. 그러나 앞의 문제 풀이를 제대로 습득하였다면, 큰 어려움 없이 풀 수 있으리라 생각된다. 정답은 맨 마지막을 참고하기 바란다.

인터뷰 문제

어떤 숫자가 2의 거듭제곱수 (2, 4, 8, 16, 32, 64, ...)인지 아닌지를 판단하는 코드를 작성하시오.

지금 소개한 문제는 정식 인터뷰에서 받은 문제가 아니라 Job Fair 같은 곳에

서 이루어진 일종의 쪽지 시험에서 나온 문제였다. 앞에서도 이미 아마존 같은 기업은 이력서를 받는 그 자리에서 C 키워드 중 volatile이 무엇을 하는지 묻기도 한다고 쓴 바가 있다. 이 문제는 실제 Job Fair에서 어떤 기업이 이력서를 받으면서 학생들에게 준 간단한 코딩 테스트이다 그러나 결코 만만치 않은 문제일 수 있다.

만약 숫자들이 컴퓨터로 어떻게 표현이 되는지에 대한 정보가 없다면 다소 까다롭고 복잡한 루틴이 나올 수 있다(예를 들어, 2로 나눈 뒤에 나머지가 있는지 확인하고, 이 과정을 몫이 1이 될 때까지 반복하면 될 것이다). 그러나 거의 모든 컴퓨터들은 정수 표현을 2의 보수로 하기 때문에 이 특징을 활용하면 훨씬 빨리 문제를 풀 수 있다. 따라서 이 문제를 풀기 전, 문제에서 주어진 숫자들은 부호와 2의 보수로 주어졌다고 가정을 해보자. 항상 문제를 풀 때 부족하다고 느껴지는 가정은 쓰는 것이 좋다. 그리고 이 가정을 세우는 것도 중요한 능력 중에 하나이다.

먼저 2의 거듭제곱수들을 2진수로 표현을 해보자. 금방 규칙 하나를 발견할 수 있을 것이다. 오직 한 자리수만 1로 되어있고 나머지는 모두 0임을 확인할 수 있다. 단순하게는 32 혹은 64번만큼 루프를 돌면서 1이 하나만 켜져 있는지 보면 될 것이다. 아니면 대부분의 CPU가 지원하는 기본 명령인 BitScanForward와 BitScanReverse를 이용하는 방법도 있을 것이다.[14] LSB에서 시작하여 MSB로 가면서 1이 발견되면 그 위치를 알려주는 함수들이다(Reverse는 MSB에서 LSB로 탐색). 그래서 이 두 함수로부터 얻은 위치가 같으면 2의 거듭제곱수라고 판단할 수 있다. 그러나 이렇게 해도 뭔가 깔끔하고 아름답지가 못하다. 더 좋은 방법이 있다.

주어진 숫자에서 1을 한 번 빼보자. 그러면 2진수 표현 방법으로는 가장 먼저 발견된 1 뒤의 모든 숫자들이 모두 1로 변할 것이다. 그리고 원래 수와 AND 연산을 한다면? 2의 거듭제곱수라면 결국 0이 얻어질 것이다. 이것이 가장 빠른 2의 거듭제곱수 판단 방법이다. 예를 들면 다음과 같다. x는 2의 거듭제곱

14. 이 두 함수에 대한 자세한 설명은 MSDN을 참고하기 바란다.

수이고, y는 그렇지 않은 수이다.

x = 0...010...0	y = 0...010...010...0
x − 1 = 0...001... 1	y − 1 = 0...010...001...1
x & (x − 1) = 0...000...0	y & (y − 1) = 0...010...000...0

이 정도면 충분히 이 원리에 대해서 이해가 갔으리라 믿는다. 그러나 여기서도 또 예외가 하나 있다. Boundary check를 해야 하는데, 0이 주어진다면 이 테스트로는 참이 얻어질 것이다. 또, 음수도 무시해야 할 것이다. 따라서 최종 정답은 (x 〉 0) && ((x & (x − 1)) == 0) 이 될 것이다. 좀 더 엄밀하게 말해 이 조건은 부호와 2의 보수로 표현된 정수 x가 2의 거듭제곱수인가와 필요충분조건을 이룬다.

인터뷰 문제

다음 매크로가 하는 일이 무엇인지 기술하시오.

```
#define foo(a, b)     ( (size_t) &((a)NULL)->b )
```

많은 프로그래밍 경험이 있다면 이 매크로가 무엇을 하는 것인지 금방 알아차릴 것이지만, 그렇지 않다면 당혹스러울 수 있다. NULL을 이상하게 캐스팅을 하고 거기에 멤버 변수까지 접근하니 "이거 버그 아니야?"라고 갸우뚱할 수 있다. 그러나 이것은 올바른 코드이고 자주 사용되는 매크로이다.

이것은 주어진 class 혹은 struct에서 특정 멤버 변수의 offset을 알아내는 것이다. 컴파일러마다 혹은 컴파일러 옵션마다 struct 자료 구조에서 멤버 변수들의 공간을 할당할 때, alignment가 다를 수가 있다. 그래서 클래스에서 어떤 멤버 변수가 얼마나 떨어져 있는가를 알고 싶다면 이 매크로를 사용하여 유용하게 알 수 있다. 이 매크로의 경우에 a에는 클래스에 대한 포인터, b에는 멤버 변수 이름을 넘겨주면 이 변수의 오프셋을 알 수 있다. 0번 주소라는 잘못된 주소를 강제로 주어진 클래스 포인터 타입으로 바꾼 뒤에 멤버 변수의 주소를 반환하는 것이다.

예제 코드를 통해 이 매크로가 어떻게 작동하는지 살펴보자.

```
struct Rectangle
{
   int top;
   int left;
   int bottom;
   int right;
};

#define foo(a, b)     ( (size_t) &((a)NULL)->b )

int offset = foo(struct Rectangle*, bottom);
```

위와 같은 코드를 컴파일하면 일반적인 경우 offset에 8이라는 숫자가 얻어짐을 확인할 수 있다(물론, 컴파일러마다 컴파일러 세팅 마다 값이 다를 수 있다).

이런 매크로를 이용한 장난은 상당히 다양한데, 그래도 C/C++을 자신있게 다룬다고 말을 할 수 있다면 간단한 매크로 구문은 충분히 숙지해야 할 것이다. 대표적으로 #define에서 사용되는 #와 ## preprocessor operator들이 있다. #는 stringizing operator라고 불리며 주어진 내용을 따옴표로 감싸 문자열로 바꿔주는 역할을 한다. ##는 Token-Pasting 혹은 merging operator라고 불리며 앞 토큰과 뒤에 있는 토큰[15]을 이어주는 역할을 한다. 좀 더 자세한 예는 검색을 통해서 쉽게 찾을 수 있을 것이다.

지금까지 8문제를 풀어보았다. 보다시피 그 자리에서 아주 높은 수준의 번뜩이는 천재성을 요구하는 문제는 없다. 모두 기본적인 전산 실력이 있으면 풀 수 있는 내용들이다.

15. 토큰(token)은 컴파일러, 파서 등에서 공백을 제외한 의미를 가지는 최소한의 문자열로 정의된다.

마지막으로 이 문제들 외에 실제 면접에서 받은 문제 몇 개를 간략하게 소개하면 다음과 같다.

① 숫자를 문자열로 바꾸는 문제(혹은 그 반대, 유니코드 관련 문제도 언급하면 좋을 것이다)

② 그래프 알고리즘 중 DFS/BFS를 응용하는 문제(반드시 이 두 가지 순회 방법은 숙지하길 바란다)

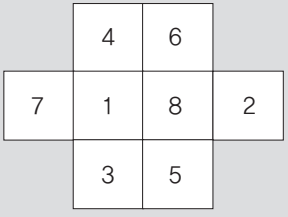
③ MODE(주어진 샘플에서 가장 많은 빈도로 나타나는 값)을 찾는 알고리즘

④ Linked List를 가지고 뒤집는 등의 연산을 하도록 하는 함수 만들기

⑤ Quicksort 알고리즘 중 partition 알고리즘을 응용한 문제

정답

• 8개의 박스에 숫자 채워 넣기 퍼즐의 정답은 아래와 같다.



• 경우의 수에 대한 답은 8!/(3!*5!)=56가지이다.

• $D(n)$의 점화식은 다음과 같다.

$D(n) = D(n-1) + D(n-2) + 2^{n-2}$, (단, n은 2보다 큰 정수이며, $D(1) = 0$, $D(2) = 1$)