**Problem1.**

첫째, 0개 이상의 입력(Input)이 존재해야 한다.

둘째, 1개 이상의 출력(Output)이 존재해야 한다.

셋째, 각 명령어의 의미는 모호하지 않고 명확해야 한다는 명확성(Definiteness)이 필요하다.

넷째, 한정된 수의 단계 후에는 반드시 종료되어야 하는 유한성(Finiteness)이라는 조건을 가져야 한다.

다섯째, 각 명령어들은 실제 실행이 가능한 혹은 계산가능한(Effective Computation) 연산이여야 한다.

**Problem2.**

프로그램을 평가하는 방법 중 실제 프로그램의 수행시간을 측정하는 방법은 동일한 하드웨어에서 사용해야 하므로 해당 문제 상황에는 적합하지 않다. 따라서 절대적인 수행 시간을 측정하는 방법이 아닌 프로그램이 수행하는 연산들의 수행 빈도수를 측정하여 프로그램이 수행되는데 시간이 얼마나 걸릴지, 얼마나 많은 공간(메모리)를 차지할지 와 같은 시간 복잡도와 공간 복잡도를 분석하는 복잡도 분석으로 직접 프로그램을 수행하지 않고도 프로그램을 평가할 수 있다.

**Problem3.**

10^12 이라는 크기를 갖는 배열을 활용하기엔 시간과 공간의 복잡도가 너무 커진다. 이렇게 대부분의 값이 0인 배열의 경우 희소행렬로 나타내는 것이 좋다. 희소행렬은 0이 아닌 값들의 행과 열의 인덱스와 실제 값만을 저장하는 배열이다. 따라서 행렬의 열은 행과 열의 인덱스를 저장하는 열 그리고 실제 0이 아닌 값을 저장하는 열로 3개뿐이고, 첫 번째 행은 희소행렬로 바꾸기 전 행렬의 행과 열 개수, 그리고 0이 아닌 값의 개수를 담는다. 이 문제의 경우 10^5 X 10^7 행렬 이므로 희소 행렬의 0번째 행의 값은 5, 7, (0이 아닌 값의 개수) 가 저장될 것이다. 따라서 이 희소행렬은 3 X (0이 아닌 값의 개수+1) 만큼의 크기만을 필요로 하기 때문에 더욱 효과적인 입력 형태를 가질 수 있다.

**Problem4.**

**(1)**

배열을 포인터 형식으로 받아 선택 정렬하는 함수이다.

배열의 크기를 n으로 받아 0부터 n-1 까지 1씩 증가시켜 n-1번 반복하면 두번째 반복문에서 인덱스가 i 보다 큰 배열의 원소 값 중 a[i] 보다 작은 값을 갖는 원소가 있는지 검사하여 그런 값이 있다면 인덱스 i를 갖는 위치에 있는 값과 교환한다.

첫번째 반복문이 반복될 때마다 인덱스 0부터 오름차순으로 정렬되고 정렬된 배열을 그때그때 출력한다.

시간의 복잡도는 O(n^2) 이다.

**(2)**

배열을 전달받고 n이라는 정수도 전달받는다.

배열에 -999보다 큰 값을 갖는 원소가 있다면 value 값을 해당 값으로 변환하고

이 과정을 배열의 인덱스 n부터 시작해 인덱스1까지 총 n번을 반복적으로 검사한 후

가장 마지막에 나오는 -999보다 큰 배열의 원소 값을 최종 value 값으로 하여 반환한다.

시간의 복잡도는 O(n) 이다.

**Problem5.**

간단히 말하자면 선입후출이라는 스택의 특징 덕분이다. 괄호 매칭 같은 경우 가장 나중에 나온 ‘(‘ 괄호와 가장 먼저 나온 ‘)’ 괄호가 짝을 이룬다. 따라서 왼쪽에서 오른쪽으로 수식을 검사하다 ‘(‘ 괄호가 나올 때마다 스택에 해당 위치를 저장하면 뒤에 나오는 ‘(‘ 괄호 일수록 스택의 top에 쌓이게 되고 이후에 ‘)’ 괄호가 나올 때마다 이것과 매칭되는 ‘(‘ 의 위치를 알고 싶다면 간단히 스택의 top을 출력하는 것으로 매칭 시킬 수 있다.

**Problem6.**

먼저 postfix 표현으로 바꾸기 위해 이를 도와줄 스택이 있다고 가정하고 왼쪽에서 오른쪽으로 수식을 검사하며 postfix 표현으로 바꿔볼 것이다.

먼저 피연산자를 만나면 바로 출력해준다.

**스택: (비어있음) postfix 표현: A**

연산자를 만나면 검사가 필요하다. 현재 스택의 top에 산술연산자가 있고 이 연산자의 우선순위가 검사가 필요한 연산자보다 우선순위가 높거나 같으면 스택의 top에 있는 연산자를 먼저 출력하고 해당 연산자를 스택에 넣는다. 그렇지 않으면 그냥 바로 스택에 넣어준다.

**스택: \* postfix 표현: A**

피연산자 B를 만났으니 그대로 출력해준다.

**스택: \* postfix 표현: AB**

/연산자를 만났으니 스택의 \*와 비교해주는데 연산 우선순위가 동일하므로 스택의 \*을 출력한다.

**스택:/ postfix 표현: AB\***

‘(‘ 괄호를 만났을 때는 ‘(‘ 를 스택에 넣어주고 넘어간다. (여기선 총 두 번을 연속해서 만난다.)

**스택:/(( postfix 표현: AB\***

피연산자 C를 만났으니 그대로 출력해준다.

**스택:/(( postfix 표현: AB\*C**

스택의 top에 연산자가 없으므로 + 연산자는 스택에 넣어주고, 그 다음 피연산자는 출력해준다.

**스택:/((+ postfix 표현: AB\*C**

**스택:/((+ postfix 표현: AB\*CD**

‘)’를 만나면 ‘(‘가 나올 때까지 스택의 top들을 하나씩 출력해준다. (출력한 스택의 값과 ‘)’와 매칭되는 ‘(‘ 는 스택에서 지운다)

**스택:/( postfix 표현: AB\*CD+**

그러면 다시 스택의 top엔 연산자가 없으므로 /를 넣어주고, E는 출력해주고, ‘)’를 만나니 위와 같은 방법을 똑같이 진행한다.

**스택:/(/ postfix 표현: AB\*CD+**

**스택:/(/ postfix 표현: AB\*CD+E**

**스택:/ postfix 표현: AB\*CD+E/**

-연산자와 /를 비교했을 때 /의 우선순위가 더 높으므로 /를 출력해주고 -는 스택에 넣는다

**스택: - postfix 표현: AB\*CD+E//**

G를 출력한 후, 수식의 검사가 다 끝난 후에 스택의 top에 남아있는 값들을 순서대로 출력해준다.

**스택: - postfix 표현: AB\*CD+E//G**

**스택: postfix 표현: AB\*CD+E//G-**

결과적으로 주어진 수식의 postfix 표현은 AB\*CD+E//G- 가 된다.

**Problem7.**

**(1)**

큐는 가장 먼저 추가된 값이 삭제될 때도 가장 먼저 삭제된다는 특징을 가지고 있다. 따라서 링크드리스트로 큐를 표현한다면 가장 먼저 들어간 노드를 가르키는 노드 포인터 (ex) NODE \*front) 가장 나중에 들어간 노드를 가르키는 노드 포인터(ex) NODE \*rear) 가 있어야 할 것이다. 만약 큐에 노드를 추가한다면 추가할 때마다 rear도 추가한 노드를 가르키게 변경하면 된다. 그리고 큐에서 삭제를 할 경우 front를 다음 노드를 가르키게 변경하고, 링크드리스트의 경우 노드를 동적으로 할당 받기 때문에 삭제할 노드는 free()로 동적메모리 반환을 해주면 된다.

**(2)**

스택은 가장 먼저 추가된 값이 가장 나중에 삭제되는 특징을 가지고 있다. 가장 나중에 들어온 노드를 가르키는 노드 포인터 top을 만들고 노드가 추가될 때마다 top 포인터로 이 노드를 가르켜주면 된다. 스택에서 값을 뺄 때도 top포인터가 하나 이전 노드를 가르키도록 변경해주고 뺄 노드는 동적 할당을 반환을 해주어 삭제 해주면 된다.