REPORT #2

|  |
| --- |
| 제 출 일 : 2019년 09월 24일 |
| 과 목 명 : 알고리즘설계와해석 |
| 담당교수 : 김희철교수님 |
| 학 과 : 컴퓨터전자시스템공학부 |
| 학 번 : 201803818 |
| 성 명 : 한아영 |

HANKUK UNIVERSITY OF FOREIGN STUDIES



1 . 지수승 계산 후 나머지 연산

2. 주식 최대 이득 찾기

3. 정렬 순서 구하기

1 . 지수승 계산 후 나머지 연산

1. Problem Definition

양의 정수 a, n, m을 입력 받고 의 값을 출력한다. (mod 는 나머지 연산자)

1. Solution

이 연산을 하기 위해 먼저 모듈러 연산의 곱셈 성질을 이해한다.

함수 calc 는 a, n, m 을 인자로 하는 재귀함수이다.

임의의 어떤 수 k의 n승을 구할 때 k를 n번 곱해도 되지만 다음을 이용하면 좀 더 효율적으로 계산할 수 있다.

|  |  |
| --- | --- |
| < n 이 짝수 일 때 > | < n 이 홀수 일 때 > |

그렇다면 을 계산하는 calc함수도 다음과 같이 구현할 수 있다.

1\_ calc함수는 n이 0일 때 1을 return 한다. (base case)

2\_ calc함수는 n이 홀수 일 때 다음 값을 return 한다.

3\_ calc함수는 n이 짝수일 때 다음 값을 temp 변수에 저장한다.

temp의 제곱을 구하여 m으로 나눈 나머지 값을 구한 뒤 return한다.

이때 와 는 calc함수로 다시 구현할 수 있다.

따라서 calc함수는

1\_ calc함수는 n이 0일 때 1을 return 한다. (base case)

2\_ calc함수는 n이 홀수 일 때 다음 값을 return 한다.

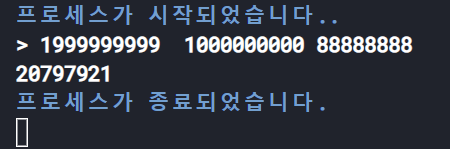
3\_ calc함수는 n이 짝수일 때 다음 값을 temp 변수에 저장한다.

temp의 제곱을 구하여 m으로 나눈 나머지 값을 구한 뒤 return한다.

거듭제곱을 하다 보니 숫자가 커지는 것을 대비해 변수를 long long으로 선언하였다.

인자로 받은 수를 제곱해서 return하는 square()함수를 구현해 temp의 제곱을 하였다.

1. Result



1. Conclusion

처음에는 단순하게 을 구해서 m으로 나눈 나머지를 구하면 되는 줄 알았다.

Solution에 기재했던 ( 을 구하는 ) 방법으로 재귀함수를 구현한 후 거듭제곱을 먼저 해결하고 그 값의 mod연산을 하도록 구현했는 데 예제 case를 돌려보니 음수가 나왔다.

생각을 해보니 int형 변수는 약 23억까지 표현할 수 있는데 거듭제곱은 수가 기하급수적으로 커지기 때문에 23 억을 훌쩍 넘을 것이라 생각하게 되었다. 하물며 예제에 있는 1999999999를 10승만 해도 약 93자리의 수가 되는데 1000000000승은 세는 것도 힘들 어마무지한 수가 될 것이다. 그러다 모듈러 연산의 곱셈 성질을 알게 되었고 무작정 거듭제곱을 한 후 나머지를 구하는 것이 아님을 알았다. 같은 동작을 하더라도 어떻게 구현하는 지에 따라 수행시간과 효율성 면에서 매우 큰 차이가 나기 때문에 코드 한 번을 짜더라도 어떻게 하면 더 효율적으로 짤 수 있는지 생각해 보는 계기가 되었다.

1. Source Code

|  |
| --- |
| #include <iostream>  using namespace std;  //인자로 받은 수를 제곱하여 return하는 함수  long long square(long long n){  return n\*n;  }  //a, n, m을 받고 a^n % m의 값을 구하는 함수  long long calc (long long a, long long n, long long m){  if (n ==1 ){  return a%m;  }  else if (n%2 == 0){//a^n = a \* a^n-1 을 이용  long long tmp = square(calc(a, n/2, m)) % m;  return tmp;  }  else if (n%2 ==1){ //a^n = { a^(n/2) }^2 을 이용  long long tmp = ((a % m) \* calc(a,n-1,m))%m;  return tmp;  }  }  int main(){  long long a , n, m;  cin >> a >> n>> m;  long long result = calc(a,n,m);  cout << result;  } |

2. 주식 최대 이득 찾기

1. Problem Definition

주식 가격이 날짜별로 n개 주어져 있을 때 얻을 수 있는 최대 이득을 구한다.

1. Solution

원리는 다음과 같다.

(분할)

먼저 **배열의 최솟값의 인덱스(이하 min)**를 기준으로 2등분한다.

재귀함수를 사용해야 하는데

주식 가격을 입력 받은 배열 전체에서 최소값을 찾는다.

최소값 오른쪽에 있는 값 중에 가장 큰 값을 찾는다.

둘의 가격차이를 변수 max에 저장한다.

최소값 왼쪽에 있는 것을 새로운 배열이라 생각하고 그 안에서 최소값을 찾는다.

그 최소값의 오른쪽에 있는 값 중에 가장 큰 값을 찾는다.

**(정복)**

2등분한 배열에서 각각 나온 이익의 최댓값을 비교하면 끝.

왼쪽 배열에서 나온 값이 오른쪽 배열에서 나온 값보다 클 때만 max를 갱신한다.

이 작업을 배열에 원소가 1개 남을 때까지 반복한다.

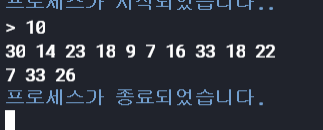
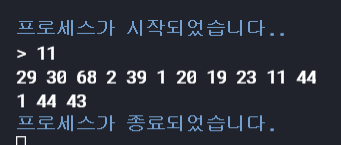
**왜 그런지 설명하자면**

재귀함수의 첫번째 cycle에서는 배열 전체의 최소값이 나오게 된다. 이때 주식을 샀다고 가정하면 앞 가격은 고려할 필요가 없으므로 뒤의 가격중 최대에서 팔면 된다.

두번째 cycle에서는 배열 전체의 두번째 최소값이 나오게 되는데 첫 번째 최소값 이후로는 어차피 첫번째 최소값일 때가 가장 큰 이익을 내므로 볼 필요가 없기에 두번째 최소값과 첫번째 최소값 사이에서 더 큰 이익이 나오는지만 살펴 max값을 갱신하면 된다. 오른쪽 배열에서 산 가격은 왼쪽 배열에서 산 가격보다 항상 낮으므로 다른 작업을 하지 않아도 같은 가격일 때 적은 돈으로 살 때의 가격이 나온다.

이것을 반복하면 배열 전체에서 가장 큰 이익을 찾을 수 있다. 사고 파는 가격은 Max값을 갱신할 때 사는 가격과 파는 가격을 변수에 대입하면 된다.

1. Result



1. Conclusion

가장 시행착오를 많이 겪었던 문제였다. 처음에는 배열의 0번째부터 마지막 인덱스 까지 for loop을 돌면서 값이 커지면 이익을 증가시키다가 최대값이 생기면 max변수를 갱신하고 값이 떨어지면 이익을 0으로 초기화하고 다시 시작하는 방법으로 구현했었는데 주식이 떨어진다고 최대이익을 낼 수 없는 것이 아니니 완전히 틀린 알고리즘이었다. 분할과 정복으로 구현해야 하는 것이 가장 난항이었다. 그러다가 배열을 2개씩 나눠서 앞에서부터 차례대로 상황을 합쳐보는 알고리즘을 구현했고 답은 항상 잘 나왔지만 그렇게 효율적인 것 같진 않았다. 그러다 최솟값을 기준으로 배열을 2등분해보자고 생각을 해보았다. 코드도 훨씬 간결해지고 무엇보다 직관적이고 간단했다. 앞서 구현했던 알고리즘은 최솟값,최댓값도 찾고 배열도 처음부터 끝까지 순회했다면 이번 알고리즘은 최솟값, 최댓값만 찾고 상황에 따라 재귀함수 한 cycle만에 결과가 끝날 수도 있었다. 가격이 계속 떨어지는 경우 (worst case)에는 시간이 가장 오래 걸리지만 가격이 계속해서 오르는 경우에는 (best case) 한 번 만에 끝난다. 수행시간은 최솟값, 최댓값을 찾아야 하므로 o(n)

1. Source Code

|  |
| --- |
| #include <iostream>  using namespace std;  class price {  public:  int buying\_price; //사는 가격  int selling\_price; //파는 가격  int profit = -1; //이익 (이익이 발생하지 않은 경우 -1 유지)  };  int find\_min(int\* arr, int first, int last); //배열에서 최솟값 찾는 function  void make\_list(int\* arr, int n);//최댓값 찾는 function  price get\_profit(int\* arr, int first, int last); //최대 이익을 찾는 function - 재귀함수  int main(){  int n;  cin>>n;  int\* list = new int[n];  make\_list(list,n);  price result;  result.buying\_price = list[0];  result.selling\_price = list[0];  result = get\_profit(list, 0, n-1);  if(result.profit!=-1)  cout << result.buying\_price<<" "<<result.selling\_price<<" "<<result.profit;  else{  cout << 0;  }  }  void make\_list(int\* arr, int n){  for (int i = 0 ; i < n ; i++){  cin >> arr[i];  }  }  int find\_min(int\*arr, int first,int last){  int min = first;  for (int i = first+1; i<last;i++){  if (arr[i]<arr[min])  min=i;  }  return min;  }  int find\_max(int\*arr, int first,int last){  int max = first;  for (int i = first+1; i<=last;i++){  if (arr[i]>arr[max])  max=i;  }  return max;  }  price get\_profit(int\* arr, int first, int last){  price current;  if (last<=0){  return current;  }  //최솟값을 기준으로 배열을 2등분한다.  //2등분한 배열에서 각각 나온 이익을 비교해서 클 때의 상황을 return한다.  int min\_index = find\_min(arr,first,last);  int max\_index = find\_max(arr,min\_index+1,last);  int max = arr[max\_index] - arr[min\_index];  price next = get\_profit(arr, first, min\_index-1);  if(first>=0){  if (max >= next.profit){  current.buying\_price= arr[min\_index];  current.selling\_price = arr[max\_index];  current.profit = max;  }  else{  current.buying\_price = next.buying\_price;  current.selling\_price = next.selling\_price;  current.profit = next.profit;  }  }  return current;  } |

3. 정렬 순서 구하기

1. Problem Definition

숫자열 R의 원소 R i 는 숫자열 s에서 원소 s i 의 보다 앞에 있는 원소 중에서 값이 작은 원소의 개수이다. R이 주어질 때 S를 찾아 출력한다.

1. Solution

숫자열 S가 { S1 , S2 , S3 , S4, … ,Sn } 이므로 이를 계산하기 편하게 하기 위해 인덱스만 리스트에 담는다. => index\_list = [1, 2, 3, …, n]로 하며 1은 S1 , 2는 S2를 의미한다.

Ri 는 Si 앞에 Si 보다 작은 값들의 개수이다. 이도 입력을 받아 배열 R\_list에 저장한다.

R[i]값에 따라 index\_list[i]값의 위치를 바꾼다.

index\_list[i]값의 인덱스를 R[i]가 되도록 한다.

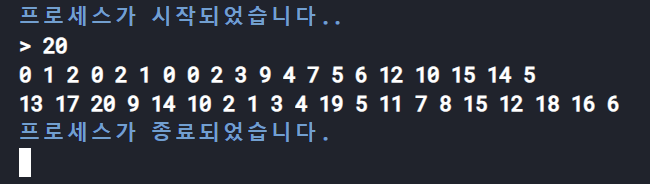
삽입을 하기 전 index\_list[0]부터 index\_list[i]까지의 길이보다 R[i]값이 크다면 숫자열 S를 찾을 수 없다.

삽입연산이 끝난 index\_list 값에 차례대로 1부터 n까지 부여한다.

가령 index\_list = [ 1, 3, 2, 5, 4] 라면 S1 = 1 , S3 = 2 , S2 = 3 , S5 = 4 , S4 = 5가 된다.

이를 index순서대로 숫자열 S\_list에 나열하면 우리가 원하는 문자열을 얻을 수 있다.

1. Result



1. Conclusion

해결 방법만 찾는 다면 구현하는 것은 쉬운 일인 것 같다. 배열을 어떻게 효율적으로 관리할 수 있는지에 대해서만 생각해보면 될 것 같다. 나는 i위치에 원소를 삽입할 때 i이후의 원소들을 하나씩 뒤로 미루고 빈 자리에 삽입했는데 더 좋은 방법이 있는지 고민해봐야 할 것 같다. 이번 3과제를 통해 더 좋은, 빠른, 효율적인 알고리즘을 찾을 수 있도록 계속 노력하는 계기가 되었다.

1. Source Code

|  |
| --- |
| #include<iostream>  using namespace std;  int get\_length(int\* list, int index);  void insert(int\* list, int num, int i );  int main(){  int n = 0;  cin >> n;  int\* R\_list = new int[n]; //숫자열 R  int\* S\_list = new int[n]; //숫자열 S  int\* index\_list = new int[n]; //숫자열 S의 인덱스list  bool exist = true; //숫자열 S를 찾을 수 없으면 false  //숫자열 R 입력받기  for (int i=0; i<n; i++){  cin >> R\_list[i];  }  //숫자열 S 입력받기  for (int i=0; i<n; i++){  index\_list[i] = i+1;  }  //R의 값에 따라 S 인덱스 움직이기  for (int i=0; i<n; i++){  if (R\_list[i] > get\_length(S\_list, i)){  exist = false;  break;  }  else{  insert(index\_list, R\_list[i] , i);  }  }  //S인덱스의 위치에 따라 값 배정  for (int i=0; i<n; i++){  S\_list[index\_list[i]-1] = i+1;  }    if(exist){  for(int i = 0 ; i < n ; i++){  cout << S\_list[i] <<" ";  }  }  else  cout<<-1;  }  int get\_length(int\* list, int index){  int count=0;  for (int i = 0 ; i < index; i++)  count++;  return count;  }  void insert(int\* list, int num, int i){  if (i>0){  int tmp = list[i];  int j = i;  int move = get\_length(list, i) - num;  for (; move > 0 ; move-- , j--){  list[j] = list[j-1];  }  list[j] = tmp;  }  } |