75 marks **CS 213(M) Quiz 1** **9:30-11:00, 25/1/18**

Problem 1:[10 marks] The function below is supposed to return the product of its arguments which are guaranteed to be positive. Let S_0, T_0 denote the initial values of `s, t`. (a) Give the main loop invariant that is needed to prove the correctness. Do not prove the invariant but merely explain why it implies that the function is correct. (b) How many iterations does the loop run? Give an estimate correct to within a constant factor in terms of S_0, T_0. Justify your answer in 3-4 lines.

```
int mult(int s, int t){
  int p=0;
  while(s > 0){
    if(s % 2 == 1){p = p + t; s = s-1;}
    else {s = s/2; t = t*2; }
  }
  return p;
}
```

(a) Loop invariant: p + st = ST, where S,T are values read earlier. [3 marks]
When the loop terminates s=0, and p + st = ST. So p = ST. [3 marks]
(b) The loop runs at most $2\log_2 S$ times. [2 marks]
This is because S halves at least once very 2 steps. [2 marks]

Problem 2:[10 marks] Write a function which takes two sorted arrays and prints out the common elements. Assume that the arrays are sorted in increasing order, i.e. no element is repeated in any single array. The function should run in time $O(m+n)$ where m, n are the lengths of the arrays. Give the main loop invariant without proof.

```
void merge(int *a, int *b, int m, int n){
    int i=0, j=0;
    while(i<m and j<n) {
        if(a[i] < b[j]) ++i;
        else if(a[i] > b[j]) ++j;
        else {
            cout << a[i] << " ";
            ++i;
            ++j;
        }
    }
}
```

```
Loop Invariant:
1. At any point of time all the common element in range a[0 ... i-1]
   and b[0 ... j-1] have been printed.
2. At any point of time there are no common elements in cross ranges
   (a[0 ... i-1] and b[j ... n-1]) or (a[i ... m-1] and b[0...j-1]).
   Only remaining common elements may be in the range a[i... m-1] and
   b[j ... n-1]

Grading Scheme:
7 marks for code:
---> If partial code is written - 3 marks.
---> Worst case complexity is more than O(m + n) - 5 marks
---> Full code with O(m + n) complexity - 7 marks
3 marks for loop invariant:
---> Something relevant is written - 1 mark
---> Point 1 of Loop Invariant (Solution) is written - 2 marks
---> Both points are written or correct answer is written in saome other form - 3 marks
```

Problem 3:[10 marks] Write a function which takes an array containing a min-heap, and a target value t and returns the number of elements in A that are smaller than t. You are expected to write a recursive function which exploits the heap ordering to do as little work as possible. Precisely state what your function does as a function of its arguments.

```
int Nsmaller(int a[], int t,int current_index, last_index){
  if(current_index > last_index || a[current_index] >=t) return 0;
    // if we are searching past the last heap element
    // or the top of the array is itself not smaller
  return 1 + Nsmaller(a,t, 2*current_index+1, last_index)
           + Nsmaller(a,t, 2*current_index+2, last_index));
    // if the root element is smaller then need to search the tree
    // return 1 for the root and for each subtree
}

// Nsmaller returns the number of elements smaller than t in the heap
// with root at a[current_index] and stored in a[0..last_index]
```

Marking scheme :-

7 marks for code out of which 3 marks for base case at which recursion stops and 4 marks for inductive step where the return statement should be correct.

3 marks for explanation which includes function argument explanation, how the recursion works and the property of min-heap we used in recursion.

Problem 4:[20 marks] Write a program that takes as input the integers N, T, and a sequence S of N integers, and prints true iff some subsequence of S adds up to T. Note that a subsequence may contain non consecutive elements of the original sequence. You are expected to go through all 2^N subsequences of S systematically and check if any adds up to T. Precisely state what the main function does as a function of its arguments.

```
Three methods were used to solve this problem.  The best one is given below.
In this we maintain the sum of the selected elements, and not the
selected elements.  This makes for simpler code.  But if you
maintain the selected elements and check whether they add up to T at
the end that is also fine (second method).  The third method was to
use the bit pattern of integers between 0 and 2^n - 1 as a
representation of n bit strings.  This will work only if n < 32 (if
you use int) or 64 (long).  If you do this you can get only upto 10 marks.
---

bool SubseqCheck (int A[], int N, int T, int current_index, int current_sum){
  if (current_index == N) return (current_sum == T)
    // if all elements have been considered, check whether sum is T

  else return (SubseqCheck(A,N,T,current_index+1,current_sum+A[current_index]) ||
               SubseqCheck(A,N,T,current_index+1,current_sum));

   // Check if we can get to T by including the current element or not.
}

int main(){
  int N,T;
  cin>>N>>T;
  int S[N];
  for (int i=0; i<N; i++) cin>>S[i];
  cout << SubseqCheck(S,N,T,0,0) << endl;
}

// This function decides if some subset of the elements in
// A[current_index..N-1] can add up to T - current_sum

------
Grading Scheme:

Base case: 5 marks

Inductive step: 10 marks
Considering both cases: a) Having the current element as part of the
subsequence, and b) Not considering the current element as part of the subsequence

Explaination of functionality: 5 marks
```

Problem 5:[25 marks] Write a function that takes an array A sorted in non-decreasing order, and a value v, and returns the index of the first occurrence of v in the array. If v does not appear in the array, anything can be returned. Your function should run in time $O(\log n)$ where n is the length of the array. Give a proof of correctness.

```
Marks division : 2+10+8+3+2
code :
1) correct base condition (2M)
2) Traversing to the correct side after checking if condition (10M)
3) Program is terminating. (8M)
Proof of correctness :
1) Proof that code is traversing to the correct direction. (3M)
2) Proof that code is terminating (2M)

int search(int *A, int i, int L, int t){
// Returns the first occurrence of t (if any) in A[i,i+L-1]

  if(L <= 1) return i;

  int Hminus = (L-1)/2;
  if(t > A[i+Hminus]) return search(A,i+Hminus+1,L-Hminus-1,t);
  else return searh(A,i, Hminus+1,t);

// Proof of correctness
// if t > A[i+Hminus], first occurrence is in A[i+Hminus+1...i+L-1]
//    so length is i+L-1 - (i+Hminus+1) + 1 = L - Hminus -1
// if t <= A[i+Hminus], first occurrence is in A[i..i+Hminus]
//    so length is i+Hminus - i + 1 = Hminus + 1
// But 0 < Hminus+1 < L for L > 1.  So length goes down in each call.
}
```