Name:

Login:

Signature:

ECE 368 Spring 2016.

Homework 2

**1) Finding Peaks in an $n \times n$ matrix (10pts).**

Consider the following divide and conquer algorithm for peak finding:

1. Look at the middle column
2. Find the maximum of this column
3. If it is a peak (larger than all 4 neighbors):
   - return it
4. Else:
   - Look at larger neighbor (left/right)
   - Go to step 1 with the (left/right) half

| | | | | | | |
|---|---|---|---|---|---|---|
| 9 | 3 | 5 | 2 | 4 | 9 | 8 |
| 7 | 2 | 5 | 1 | 4 | 0 | 3 |
| 9 | 8 | 9 | 3 | 2 | 4 | 8 |
| 7 | 6 | 3 | 1 | 3 | 2 | 3 |
| 9 | 0 | 6 | 0 | 4 | 6 | 4 |
| 8 | 9 | 8 | 0 | 5 | 3 | 0 |
| 2 | 1 | 2 | 1 | 1 | 1 | 1 |

This algorithm needs $O(\lg n)$ iterations and $O(n)$ to find the max in the column per iteration. Thus, the complexity of this algorithm is $O(n \lg n)$.

However, there is a more efficient peak finding algorithm. The algorithms is as follows:

1. Look at the middle row and column, and the boundaries
2. Find the maximum within these rows/columns
3. If it is a peak (larger than all 4 neighbors):
   - return element
4. Else:
   - Look at larger neighbor
   - Go to step 1 with the quadrant containing the larger neighbor.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 9 | 3 | 5 | 2 | 4 | 9 | 8 | 0 |
| 0 | 7 | 2 | 5 | 1 | 4 | 0 | 3 | 0 |
| 0 | 9 | 8 | 9 | 3 | 2 | 4 | 8 | 0 |
| 0 | 7 | 6 | 3 | 1 | 3 | 2 | 3 | 0 |
| 0 | 9 | 0 | 6 | 0 | 4 | 6 | 4 | 0 |
| 0 | 8 | 9 | 8 | 0 | 5 | 3 | 0 | 0 |
| 0 | 2 | 1 | 2 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*Prove that the second algorithm works (it always finds a peak) and derive its worst-case time complexity.*

2) **Greedy Ascent Algorithm (10 pts).**

The greedy ascent peak finder algorithm starts from the element in the middle column and middle row (assuming a square matrix), and then compares the element with neighboring elements in the following order (left, down, right, up). If it finds a larger element, then it moves to that element. Eventually, it is guaranteed to stop at a peak that is greater than or equal to all its neighbors. In class, we discussed an example where greedy ascent stops after stopping by 20 elements. You are required in the exercise to come up with a 5x5 matrix where greedy ascent will stop at the 23rd element.

| | | | | |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

3) **Stacks and Queues (10 pts):**

One common interview question is to show how to create a stack from other abstract data types(ADT). Your task is to write pseudocode to implement a stack using two queues with the following primitives of queues: EMPTY(Q), ENQUEUE (Q, element), DEQUEUE(Q). Your stack should have the following primitives: *PUSH (S, element) and POP(S).* What is the time complexity for each of the operations PUSH (s, element) and POP(s)?

## 4) Analysis of algorithms (10 pts):

| INSERTION_SORT($A[1..n]$) | *Cost* | *Times* |
|---|---|---|
| 1. for $j \leftarrow 2$ to $n$ | $C_1$ | |
| 2. key $\leftarrow A[j]$ | $C_2$ | |
| 3. $i \leftarrow j-1$ | $C_3$ | |
| 4. while $i > 0$ and $A[i] > key$ | $C_4$ | |
| 5. $A[i+1] \leftarrow A[i]$ | $C_5$ | |
| 6. $i \leftarrow i-1$ | $C_6$ | |
| 7. $A[i+1] \leftarrow$ key | $C_7$ | |

(a) Let $t_j$ denote the number of times the while loop test in line 4 is executed for that value of $j$. Fill in for each line of instruction, the number of times the instruction is executed.

(b) Derive the expression for the running time of INSERTION_SORT in terms of $n$, $C_i$, and $t_j$.

(c) What is $t_j$ for the best-case scenario, i.e., when the running time of the algorithm is the smallest. Use that to derive the expression for the best-case running time of INSERTION_SORT in terms of $n$ and $C_i$. What is the best-case time complexity of INSERTION_SORT using the big-O notation?

(d) What is $t_j$ for the worst-case scenario, i.e., when the running time of the algorithm is the largest. Use that to derive the expression for the worst-case running time of INSERTION_SORT in terms of $n$ and $C_i$. What is the worst-case time complexity of INSERTION_SORT using the big-O notation?

## 5) Recursive Algorithms (10 pts):

The following C++ function *permute()* prints all permutations of the given string. For example, a call of *permute(0,2)* on "ABC" should print the following (order does not matter).

ABC ACB BAC BCA CBA CAB

*Complete the following code and briefly explain what happens to the string in every recursive call. (Note: you shouldn't need to write more than 5 lines of code*).

```
#include <iostream>
using namespace std;

char str[] = "ABC";

void swap (char *x, char *y){
    char temp;
    temp = *x;
    *x = *y;
    *y = temp;
}

void permute(int i, int n){
  int j;
  if (i==n){
      cout << str << " ";
  }else{
      for(j=i;j<=n;j++)
      {
        //Your code goes here




      }
    }
}
```