Name: Homework Solutions.
Login:
Signature:

ECE 368 Spring 2016.

Homework 2

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

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

1. **Correctness:**
   - If you enter a quadrant, this means that the maximum element on the border of the quadrant is not a peak.
   - When the algorithm stops inside a quadrant, the peak is internal to the quadrant so it is a peak in the array.

2. **Complexity:**
   1. Given $n \times n$ matrix, you will look into middle row($n$), middle column($n$) and borders ($c \times n$)
      \[= n + n + cn.\]
   2. Then call function recursively in $\frac{n}{2} \times \frac{n}{2}$ matrix. = middle row ($\frac{n}{2}$), middle column($\frac{n}{2}$) and new borders ($c \times \frac{n}{2}$).
   3. Total complexity of function is:
      - $F(n) = F\left(\frac{n}{2}\right) + cn$.
      - $F\left(\frac{n}{2}\right) = F\left(\frac{n}{4}\right) + c\frac{n}{2}$.
      - Writing in terms of $F(n)$: $F(n) = F(1) + c\left(2 + 4 + \ldots + \frac{n}{4} + \frac{n}{2} + n\right)$
      - Largest term is $n$, so function is $O(n)$.
Greedy Ascend Algorithm (10 pts).
Describe the greedy ascend algorithm and come up with a 5x5 matrix where greedy ascend will stop at the 23erd element.

<table>
<thead>
<tr>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

2) 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)?

### Push(s, e)
ENQUEUE(Q1, e)

### Pop(s)
if EMPTY(Q1) (ALWAYS CHECK)  
return error
while not EMPTY(Q1)
    temp = pop(Q1);
    if EMPTY(Q1) (temp = last element)  
        ENQUEUE all elements from Q2 to Q1
    return temp
else
    ENQUEUE(Q2, temp)
Complexity: Push = O(1), POP = O(n)

### Solution 2 (POP efficient):

### Push(s,e)
ENQUEUE(Q2, e)

ENQUEUE all elements from Q1 to Q2
ENQUEUE all elements from Q2 to Q1

### Pop(s)
if EMPTY(Q1) (ALWAYS CHECK)  
return error
else
    return DEQUEUE(Q1)
Complexity: Push = O(n), POP = O(1)
3) Analysis of algorithms (10 pts):

```
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Cost</th>
<th>Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>for j ← 2 to n</td>
<td>C_1</td>
<td>n</td>
</tr>
<tr>
<td>key ← A[j]</td>
<td>C_2</td>
<td>n - 1</td>
</tr>
<tr>
<td>i ← j - 1</td>
<td>C_3</td>
<td>n - 1</td>
</tr>
<tr>
<td>while i &gt; 0 and A[i] &gt; key</td>
<td>C_4</td>
<td>∑_{j=2}^{n} t_j</td>
</tr>
<tr>
<td>A[i + 1] ← A[i]</td>
<td>C_5</td>
<td>∑_{j=2}^{n} (t_j - 1)</td>
</tr>
<tr>
<td>i ← i - 1</td>
<td>C_6</td>
<td>∑_{j=2}^{n} (t_j - 1)</td>
</tr>
<tr>
<td>A[i + 1] ← key</td>
<td>C_7</td>
<td>n - 1</td>
</tr>
</tbody>
</table>
```

(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 \( t_j \).

Let \( T(n) = \text{running time of INSERTION_SORT} \).

\[
T(n) = C_1 n + C_2 (n - 1) + C_3 (n - 1) + C_4 \sum_{j=2}^{n} t_j + C_5 \sum_{j=2}^{n} (t_j - 1) + C_6 \sum_{j=2}^{n} (t_j - 1) + C_7 (n - 1)
\]

(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?

The array is already sorted. All \( t_j \) are 1. Therefore,

\[
T(n) = C_1 n + C_2 (n - 1) + C_3 (n - 1) + C_4 (n - 1) + C_7 (n - 1)
= (C_1 + C_2 + C_3 + C_4 + C_7)n - (C_2 + C_3 + C_4 + C_7).
\]

\( T(n) = O(n) \). (In fact, \( T(n) = \Theta(n) \).)

(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?

The array is in reverse sorted order. We have to compare the \( j \)-th element with the previous \( (j - 1) \) elements. We also need an additional test to get out of the while-loop. Therefore, \( t_j = j \). Hence,

\[
T(n) = C_1 n + C_2 (n - 1) + C_3 (n - 1) + C_4 \sum_{j=2}^{n} j + C_5 \sum_{j=2}^{n} (j - 1) + C_6 \sum_{j=2}^{n} (j - 1) + C_7 (n - 1)
= C_1 n + C_2 (n - 1) + C_3 (n - 1) + C_4 \left( \frac{n(n+1)}{2} - 1 \right) + C_5 \frac{n(n-1)}{2} + C_6 \frac{n(n-1)}{2} + C_7 (n - 1)
= \left( \frac{C_4 + C_5 + C_6}{2} \right) n^2 + \left( C_1 + C_2 + C_3 + \frac{C_4 - C_5 - C_6}{2} + C_7 \right) n - (C_2 + C_3 + C_4 + C_7).
\]

\( T(n) = O(n^2) \). (In fact, \( T(n) = \Theta(n^2) \).)
4) 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).

```cpp
#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
            swap((str + i), (str+j));
            permute(i+1, n);
            swap((str+i),(str+j));
        }
    }
}
```