Quiz 2

22C:80 Programming for Informatics
Tuesday, April 7th, 2009

Notes: The quiz is worth **6 points total**. To get partial credit you must show your work.

---

1. Consider the following *binary search* function:

```
def binarySearch(C, x):
 first = 0
 last = len(C)-1

 while(first <= last):
  mid = (first + last)/2
  if(x == C[mid]):
   return True    # 1b) return mid

  if(x < C[mid]):
   last = mid - 1
  else:
   first = mid + 1

 #The sublist has shrunk to size 0 and so x is not in the list
 return False     # 1b) return first
```

   (a) Suppose that `C` is `[3, 5, 5, 11, 20, 21, 50, 57]`. How many times will the body of the while-loop execute when `binarySearch(C, 12)` is called?

   Solution: 3 times
   1st time: $f = 0, l = 7, m = 3$
   2nd time: $f = 4, l = 7, m = 5$
   3rd time: $f = 4, l = 4, m = 4$

   (b) Sometimes `binarySearch` is more useful if it returns an index, rather than just a `True` or `False`. Modify `binarySearch` so that (i) if `x` is found in `C`, the function should return the index of a slot that `x` is in; (ii) otherwise, it should return the index of the slot that `x` would inserted into, if `C` were to remain sorted after the insertion. For example, `binarySearch(C, 12)` should return 4 because if 12 were to be inserted into `C`, it would go into the slot currently occupied by 20, and the index of that slot is 4. You don't have to rewrite `binarySearch`, just list the modifications you will make.

Solution:

Modifications noted in the code above (as comments).

2. The code for the functions `partition` and `recursiveQuickSort` are given below:

```
def partition(C, left, right):
  p = left
  for i in range(p+1, right+1):
    if(C[i] < C[p]):
      swap(C, i, p+1)
      swap(C, p, p+1)
      p = p + 1

  return p

def recursiveQuickSort(C, left, right):
  if(left < right):
    p = partition(C, left, right)
    recursiveQuickSort(C, left, p-1)
    recursiveQuickSort(C, p+1, right)
```

(a) Suppose `C` is the list `[11, 7, 2, 4, 8, 1, 17, 5]` and you call `partition(C, 1, 5)`. How does `C` change as a result of this call? What is the value returned by `partition(C, 1, 5)`?

Solution: $[11, 2, 4, 1, 7, 8, 17, 5]$
The value returned by partition$(C, 1, 5)$ is **4**

[The values of C after iteration i are:
i=2 C=$[11, 2, 7, 4, 8, 1, 17, 5]$
i=3 C=$[11, 2, 4, 7, 8, 1, 17, 5]$
i=4 C=$[11, 2, 4, 7, 8, 1, 17, 5]$
i=5 C=$[11, 2, 4, 1, 7, 8, 17, 5]$]

(b) Suppose that we inserted a "print statement" that prints (in one line) `C`, `left`, and `right`, just before the call to the `partition` function in `recursiveQuickSort`. What are the first three lines of output you will see when you call `recursiveQuickSort(C, 0, 7)`, with `C` being the list `[11, 7, 2, 4, 8, 1, 17, 5]`.

Solution: [

C	left	right
$[11, 7, 2, 4, 8, 1, 17, 5]$	0	7
$[7, 2, 4, 8, 1, 5, 11, 17]$	0	5
$[2, 4, 1, 5, 7, 8, 11, 17]$	0	3

]

3. Consider the variant of *job scheduling* problem in which you have two conference rooms, instead of just one. You still want to maximize the total number of accepted requests, since you get paid a fixed amount per request. One natural algorithm for this variant is the following. Let I be the input set of requests. Consider one room (picked arbitrarily) and choose from I the maximum possible number of non-overlapping requests, to schedule in that room. Let S be the set of requests scheduled in the first room. Then take the unscheduled requests, i.e., $I - S$, and choose from $I - S$ the maximum possible number of non-overlapping requests to schedule into the second room. Note that you can use the "ends first" algorithm to solve the subproblem for each room.

   (a) Run the above algorithm for the input

   `[3, 7], [2, 9], [4, 8], [5, 7], [6, 11], [8, 10], [9, 15], [11, 17]`

   Clearly, show the requests that are scheduled in each room and also the unscheduled intervals. Recall our convention that intervals such as `[4, 8]` and `[8, 10]` do not overlap. **Solution:**

   Room 1: $[3, 7], [8, 10], [11, 17]$
   Room 2: $[5, 7], [9, 15]$
   [We sum the "ends first" algorithm for Room 1 & then sum the "ends first" algorithm on the unscheduled intervals for Room 2.]

   (b) Do you think this algorithm always yields an optimal solution? If so, justify your claim with an informal 1-2 sentence argument. If not, construct a counterexample showing that it is possible to do better than the algorithm described above.

   Solution:
   A: [0,3]
   B: [6,7]
   C: [1,5]
   D: [4,8]
   Consider the above input. The "ends first" algorithm will schedule intervals A & B in one room and then can only schedule C in the other room.
   A better solution is to schedul A & D in the 1st room and B & C in the 2nd room.