Homework 3
CS 457

Damien Prieur

Question 1

Prove tight **worst-case** asymptotic upper bounds for the following recurrence equation that depends on a variable $q \in [0, n/4]$. Note that you need to prove an upper bound that is true for every value of $q \in [0, n/4]$ and a matching lower bound for a specific value of $q \in [0, n/4]$ of your choosing. Do not assume that a specific q yields the worst case input; instead, formally identify the q which maximizes the running time. (Hint: look at the bottom of Page 180 for the analysis of the worst-case running time of Quicksort)

$$T(n) = \begin{cases} 1 & \text{if } n \leq 2 \\ T(n-2q-1) + T(3q/2) + T(q/2) + \Theta(1) & \text{otherwise.} \end{cases}$$

Proceed via the substitution method with $T(n) \in \Theta(n)$
Assume that $T(n') \leq cn' \quad \forall n' < n$ which gives us

$$T(n) = \max_{0 \leq q \leq \frac{n}{4}} T(n-2q-1) + T(3q/2) + T(q/2) + \Theta(1)$$

$$T(n) \leq \max_{0 \leq q \leq \frac{n}{4}} c(n-2q-1) + c(3q/2) + c(q/2) + k$$

$$T(n) \leq \max_{0 \leq q \leq \frac{n}{4}} c(n-2q-1+2q) + k$$

$$T(n) \leq \max_{0 \leq q \leq \frac{n}{4}} c(n-1) + k$$

$$T(n) \leq cn + (k-c) \leq cn$$

This inequality holds if $c \geq k$ so we have shown that

$$T(n) \in O(n)$$

Now to show $T(n) \in \Omega(n)$
Assume that $T(n') \geq cn' \quad \forall n' < n$ which gives us

$$T(n) = \max_{0 \leq q \leq \frac{n}{4}} T(n-2q-1) + T(3q/2) + T(q/2) + \Theta(1)$$

$$T(n) \geq \max_{0 \leq q \leq \frac{n}{4}} c(n-2q-1) + c(3q/2) + c(q/2) + k$$

$$T(n) \geq \max_{0 \leq q \leq \frac{n}{4}} c(n-2q-1+2q) + k$$

$$T(n) \geq \max_{0 \leq q \leq \frac{n}{4}} c(n-1) + k$$

$$T(n) \geq cn + (k-c) \geq cn$$

This inequality holds if $c \leq k$ so we have shown that

$$T(n) \in \Omega(n)$$

Therefore

$$T(n) \in \Theta(n)$$

Question 2

Given an array S of n distinct numbers provide $O(n)$-time algorithms for the following:

- Given two integers $k, \ell \in \{1, \ldots, n\}$ such that $k \leq \ell$, find all the ith order statistics of S for *every* $i \in \{k, \ldots, \ell\}$.

```
function k_l_orderStatistics(array, start, end, k, l):
    Select(array, start, end, k);

    //now everything less than k is to the left

    Select(array, k, end, l - k);

    //now everything less than l is to the left
    // all elements between k and l are k \leq a[i] \leq l
    // if we want them in order then we
    // can sort the sub array which will be of size k-l

    quicksort(array, k, l)

    // if we don't care if they are sorted we can just return and the
    // indicies of k and l will be the
    // indexes of the array holding the statistics
```

- Given some integer $k \in \{1, \ldots, n\}$, find the k numbers in S whose *values* are closest to that of the median of S.

```
function find_k_closest_to_median(array, start, end, k):
    k_closest := []
    middle := (start - end)/2
    Select(array, start, end, middle)
    median := array[middle]

    Select(array, start, middle - 1, middle - k)
    Select(array, middle + 1, end, middle + k)
    quicksort(array, middle - k, middle + k)

    // We now have the median surrounded by the
    // nearest k elements smaller and k elements larger

    i := middle - 1
    j := middle + 1

    while len(k_closest) != k:
        if(median - array[i] > array[j] - array[j]):
            k_closest.append(array[j])
            j := j + 1
        else:
            k_closest.append(array[i])
            i := i - 1
    return k_closest
```

Question 3

Consider the following silly randomized variant of binary search. You are given a sorted array A of n integers and the integer v that you are searching for is chosen uniformly at random from A. Then, instead of comparing v to the value in the middle of the array, the randomized binary search variant chooses a random number r from 1 to n and it compares v with $A[r]$. Depending on whether v is larger or smaller, this process is repeated recursively on the left sub-array or the right sub-array, until the location of v is found. Prove a tight bound on the expected running time of this algorithm.

```
function binary_random(array, start, end, v)
    r := random(start, end)
    if array[r] == v:
        return r
    if array[r] < v:
        return binary_random(array, start, r-1, v)
    return binary_random(array, r+1, end, v)
```

$$X_i = \{\text{the } i\text{'th element is selected and v is less than a[i]}\}$$

$$X_j = \{\text{the } j\text{'th element is selected and v is greater than a[j]}\}$$

$$T(n) = \sum_{i=1}^{n} X_i(T(i) + \Theta(1)) + \sum_{j=0}^{n-1} X_j(T(n-j) + \Theta(1))$$

$$E[T(n)] = E[\sum_{i=1}^{n} X_i(T(i) + \Theta(1)) + \sum_{j=0}^{n-1} X_j(T(n-j) + \Theta(1))]$$

$$E[T(n)] = E[\sum_{i=1}^{n} X_i(T(i)] + \Theta(1)) + E[\sum_{j=0}^{n-1} X_j(T(n-j) + \Theta(1))]$$

$$E[T(n)] = \sum_{i=1}^{n} E[X_i](T(i) + \Theta(1)) + \sum_{j=0}^{n-1} E[X_j](T(n-j) + \Theta(1))$$

$$E[T(n)] = \sum_{i=1}^{n} \frac{1}{n} \cdot \frac{i}{n}(T(i) + \Theta(1)) + \sum_{j=0}^{n-1} \frac{1}{n} \cdot \frac{n-j}{n}(T(n-j) + \Theta(1))$$

$$E[T(n)] = \frac{1}{n}(\sum_{i=1}^{n} \frac{i}{n}T(i) + kn + \sum_{j=0}^{n-1} \frac{n-j}{n}T(n-j) + kn)$$

$$E[T(n)] = \frac{1}{n}(\sum_{i=1}^{n} \frac{i}{n}T(i) + \sum_{j=1}^{n} \frac{j}{n}T(j) + 2kn)$$

$$E[T(n)] = \frac{2}{n^2}\sum_{i=1}^{n} iT(i) + 2k$$

Guess $T(n) \in \Theta(\log(n))$
Assume that $T(n') \leq c \log n' \quad \forall n' < n$ which gives us

$$E[T(n)] \leq \frac{2}{n^2}\sum_{i=1}^{n} ic\log(i) + 2k$$

$$\sum_{i=1}^{n} i\log(i) \leq \int_1^n x\log(x)\,dx = \frac{1}{4}(n^2(2\log(n) - 1) + 1)$$

$$E[T(n)] \leq \frac{2c}{n^2}\sum_{i=1}^{n} i\log(i) + 2k \leq \frac{2}{n^2}\frac{1}{4}(n^2(2\log(n) - 1) + 1) + 2k$$

$$E[T(n)] \leq c\log(n) - \frac{c}{2} + \frac{c}{n^2}) + 2k \leq c\log(n)$$

$$E[T(n)] \leq c\log(n) - \frac{c}{2} + \frac{c}{n^2}) + 2k \leq c\log(n)$$

For sufficiently large n we can show that

$$4k \leq c$$

Let $c = 8k$

$$T(n) \in O(\log n)$$

Guess $T(n) \in \Omega(\log(n))$
Assume that $T(n') \geq c \log n' \quad \forall n' < n$ which gives us

$$E[T(n)] \geq \frac{2}{n^2} \sum_{i=1}^{n} ic \log(i) + 2k$$

$$\sum_{i=1}^{n} i \log(i) \in \Theta(n^2 \log(n))$$

$$E[T(n)] \geq \frac{2}{n^2} cn^2 \log(n) + 2k \geq c \log(n)$$

$$E[T(n)] \geq 2c \log(n) + 2k \geq c \log(n)$$

This is true $\forall c$ so

$$T(n) \in \Omega(\log n)$$

Therefore

$$T(n) \in \Theta(\log n)$$

Question 4

You are given a set S of n integers, as well as one more integer v.

- Design an algorithm that determines whether or not there exist two distinct elements $x, y \in S$ such that $x + y = v$. Your algorithm should run in time $O(n \log n)$, and it should return (x, y) if such elements exist and (NIL, NIL) otherwise.
- Formally explain why your algorithm runs in $O(n \log n)$ time.

```
function find-distinct-sum(array, start, end, sum):
    array.sort()
    i := start
    j := end
    while i<j:
        if(array[i] + array[j] == sum):
            return (i, j)
        if(array[i] + array[j] > sum):
            j := j -1
        else:
            i := i-1
    return (NIL, NIL)
```

Where *array.sort*() is done using quicksort or mergesort which we have shown to be $\Theta(n \log(n))$ In the worst case nothing is found and i and j will meet at some index, when that happens i and j have traveresed the entire list. While traversing they only do constant work through comparisons or sums so we have:

$$T(n) = \Theta(n \log(n)) + cn$$

$$n \in O(n \log(n)) \implies \Theta(n \log(n)) + cn \in \Theta(n \log(n))$$

$$T(n) \in \Theta(n \log(n))$$

Question 5

Suppose that you are given a sorted array A of *distinct* integers $\{a_1, a_2, \ldots, a_n\}$, drawn from 1 to m, where $m > n$.

- Give an $O(\log n)$ algorithm to find an integer from $[1, m]$ that is not present in A. For full credit, find the smallest such integer.
- Formally explain why your algorithm runs in $O(\log n)$ time.

```
function smallest_missing(array, start, end):
    middle := floor((end-start)/2)
    if(array[middle] == middle + 1):
    //plus 1 for 0 indexed and numbers being drawn
    //from 1 to m so element 1 at index 0 should be 1
        return smallest_missing(array, middle, end)
    if(array[middle-1] == middle):
        return array[middle-1] + 1
    return smallest_missing(array, start, middle)
```

Since this is a recursive function we can look at the reccurence relation describing it

$$T(n) = T(n/2) + \Theta(1)$$

Each time we call the function we pass in half of the current range which gives us the $T(\frac{n}{2})$. The rest of the things that happen each call are all constant time, comparisons and some math so $\Theta(1)$
To solve we can use the master theorem with

$$a = 1 \qquad b = 2 \qquad f(n) = 1 \qquad \log_2(1) = 0$$

We can apply case 2 of the mater theorem as

$$f(n) \in \Theta(n^{log_2 1}) \implies 1 \in \Theta(1)$$

So by by master theorem case 2 we have

$$T(n) \in \Theta(\log n)$$