CSCI 3104
Spring 2018
Problem Set 4

Allison, George
02/17

February 15, 2018

1. a) The recurrence relation can be expressed in three parts:
   $T_1(n) = 2T_2(\frac{n}{2}) + \Theta(n)$ $T_2(n) = 2T_3(\frac{n}{2}) + \Theta(n)$ $T_3(n) = T_1(n-1) + \Theta(n)$
   $T_3(n)$ represents the levels of the recursion tree that choose the worst possible pivot.
   This partition algorithm can benefit or degrade the running time of QuickSort depending on how sorted the input is.

2. a) The following rows shows the remaining wands after comparisons are made. Each row has half the amount of wands as the previous row:

$$\frac{3}{2} \quad \frac{5}{2} \quad \frac{1}{2} \quad \frac{2}{2} \quad \frac{4}{2} \quad \frac{2.5}{2} \quad \frac{0.5}{2} \quad \frac{4.5}{2}$$

$$\frac{5}{2} \quad \frac{2}{2} \quad \frac{4}{2} \quad \frac{4.5}{2}$$

$$\frac{5}{2} \quad \frac{4.5}{2}$$

$$\frac{5}{2}$$

b) Every set of comparisons divides the set into half its original size since one of every two compared wands is removed (no two wands are exactly equal). This means that the amount of comparisons made is $\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + ...$ resulting in a total of $n - 1$ comparisons every time.

c) In order to find the second most powerful wand the most powerful wand must be found, then the set of all wands that were compared with the most powerful wand needs to be searched to find the most powerful wand in that subset. For example, in (a) the subset would be $\frac{3}{2}$, $\frac{2}{2}$, and $\frac{4.5}{2}$; all wands that were once compared with $\frac{5}{2}$. Since the second most powerful wand must have been compared with the most powerful at some point during the process, it will be found in that subset.

d) In part (a) the wands with power $\frac{3}{2}$, $\frac{2}{2}$, and $\frac{4.5}{2}$ will be compared, correctly determining $\frac{4.5}{2}$ to be the second most powerful wand.

3. a) The code below sorts the array in time $O(nklog(k))$ because it sorts the array by iterating through k-sized subsets:

```
mySort(Array A, int k){
   for(int i = 0; i < A.length() - k; i++)
      sort(A, i, i+k)
}
```

b) The code below sorts the array in time $O(nk)$ because it moves the smallest value to the front of the array, iterating k times for all values of n:

```
mySort(Array A, int k)
   int min = 0;
   int temp;
   for(int i = 0; i < A.length() - k; i++){ //O(n) time complexity
      for(int x=0; x < k; x++){ //O(k) time complexity
         //finding the minimum value and moving it to the front of the array;
         if(A[i+x] < min)
            temp = A[i];;
            A[i] = A[i+x];
            A[i+x] = temp;//swapped variables
      }
   }
   sort(A, A.length()-k, A.length);//this is needed because the sort will not work after
       A.length() - k, it will cause an out of bounds error
}
```

c) Code below:

```
binHeapSort(Array A, int k)
   for(int i = (k/2 - 1); i >= 0; i--){
      int max = i;
      int l = 2 * (i+1);
      int r = 2 * (i+2);
      if(l < n && A[l] > A[max])
         max = l;
      if(r < n && A[r] > A[max])
         max = r;
      k = max;
   }
   for(int i = n - 1; i >= 0; i--){
      int temp = A[i];
      A[i] = A[0];
      A[0] = temp; //swap variables
   }
}
```

d) Verbatim from wikipedia, "Once, Alex was given several different colored blocks (two red, three blue, and four greensimilar to the picture above). Pepperberg asked him, "What color three?"

expecting him to say blue. However, as Alex had been asked this question before, he seemed to have become bored. He answered "five!" This kept occurring until Pepperberg said "Fine, what color five?" Alex replied "none". This was said to suggest that parrots, like children, get bored. Sometimes, Alex answered the questions incorrectly, despite knowing the correct answer"

4) a) Finding the median takes a constant time (in the RAM computational model), and sorting $2\lfloor n^{\frac{1}{3}} \rfloor$ elements takes $\Theta(n)$ time, so both operations take a total of $\Theta(n)$ time.

b) The two partitions of array A will be separated by pivot M, the median of the random selection of $2\lfloor n^{\frac{1}{3}} \rfloor$ elements. That means that the smaller partition of the two will be composed of at least $n^{\frac{1}{3}}$, ot it will have, at most, $2 - n^{\frac{1}{3}}$ elements.

c) If $n \leq 24T(n) = O(n)$, if $n > 24T(n) = T(n - n^{\frac{1}{3}}) + T(n^{\frac{1}{3}}) + n)$

5) Code below:

```
int insertionSort(int A[], int n){
   int j, k; //two atomic operations
   int t = 0; //counter variable, not counted
   t += 3; //including the "i = 0" on the next line
   for(int i = 0 /*one operation*/; i < n /*one operation/loop*/; i++/*one operation/loop*/)
      k = A[j]; /*one operation each loop*/
      j = i - 1; /*one operation each loop*/
      t += 4;
      while(j > 0 /*one operation/loop*/ && /*one operation/loop*/ A[j] > k /*two
          operations/loop*/){
         A[j+1] = A[j]; /*four operations/loop*/
         j = j - 1; /*two operations/loop*/
         t += 10;
      }
      A[j + 1] = k; //three operations
      t += 3;
   }
   return t;
}

int* randomArray(int n){
   for(int i = 0; i < n; i++){
      A[i] = rand() % n; //random number in range of 0 to n
   }
   return A;
}
```