Henry Daniels-Koch
October 21 2016

Algorithms: Lab 5

1). We implement select by first partitioning the array to get a rank q. We then check if the index q is the index we are looking for. If so, we return the element at that index. If it is less than the this index, we recursively call select on the left side of the pivot. Otherwise, we recursively call select on the right side of the array. In the worst case scenario, we get bad partitions and get a runtime of $O(n^2)$. However, we expect to get an average runtime of $O(n)$ due to the fact that we are using recursion. To obtain an expected linear runtime, we could use a randomized partition or implement select by splitting the array up into groups of 5, finding the median of each group, finding the median of medians, and then use this median as the index for partitioning. We would then recursively call select after this partitioning.

2). To determine whether any item occurs more than $n/2$ times, we need to find the median of the list. To find the median, we call select on the $n/2$, middle index in $O(n)$ time. We then iterate through the array and count the number of elements that are the same as the median in $O(n)$ time. Thus the total runtime is $O(n)$. If more than half of the elements are the same in this list, then the elements that are the same MUST include the median. The pseduocode is as follows:

```
halfSame(A)
    median = select(A,0,n-1,n/2)
    for i=0 to n-1:
        if(A[i] = A[median]): count++
    if (count >= n/2)
        return true
```

3). To find the $\sqrt{n}$ elements that are closest to the median of the sequence, we first find the upper and lower bounds of the elements we are looking for by calling select on $n/2+\sqrt{n}/2$ and $n/2-\sqrt{n}/2$. We then iterate through the whole list and find all the elements between these two bounds. Since select is $O(n)$, and iterating through the list is $O(k)$, the total runtime is $O(n)$. The pseudocode is as follows:

```
rootMedians(array S, int n)
    max = select(S, 0, n-1, n/2 + √n/2)
    min = select(S, 0, n-1, n/2 - √n/2)
    for i=0 to i=n-1:
        if (S[i] =< max and S[i] >= min)
            print S[i]
```

4a) Given an integer k, we aim to find k elements that have rank n/k, $2n/k$, $3n/k$, ... Using a naive algorithm with repeated selection, we iterate from 0 to k and call select to find the in/k'th smallest element. This has an overall runtime of $O(nk)$

```
findRank(A, k)
    for i=0 to i=k
        rank = select(A,in/k)
        print rank
```

b) To improve the algorithm's runtime we recursively call findRank on smaller and smaller arrays. Our base case occurs when the array is smaller than the difference between two desired ranks (n/k). We call select on the new array to look for the median. We then recursively call findRank on both the bottom and top halves of the array. We know the desired ranks must be within these sections of the array since all numbers to the right of the median are partitioned to be higher than the median and all the numbers to the left of the median are partitioned to be lower than the median. Since select runs in $O(n)$ and calling findRank recursively implies $O(lg(k))$, the

total runtime is $O(nlg(k))$. The pseudocode is as follows:

```
findRank(A,p,r)
    //base case
    if((r-p)<n/k): return
    select(A,p,r,(r-p)/2)
    findRank(A,p,(r-p)/2)
    findRank(A,(r-p)/2 + 1,r)
```