Homework 3

Yousef Suleiman | Due: Feb 20

Question 1

(a)

Using an inductive proof:

- considering a base case where `length(A) == 2`, the `while` loop will execute once and `A` will need to be sorted if `q` (one of its 2 elements is put in sorted order)
- now assume that `QUICKSORT'` works for some length `n - 1` of array `A`
    - given a larger length `n` of array `A`, observe that `PARTITION` will choose some `q` such that subarray from `p` to `q - 1` will need to have a length `⩽ n`
    - thus `QUICKSORT'` will sort the leftmost subarray
    - the rightmost subarray (which will also have a length `⩽ n`) will re-enter the `while` loop and will also be partitioned and its leftmost subarray will be sorted again
    - this repeats until the length of this rightmost subarray is 1 which is alway sorted

(b)

In the worst case, the partition `q` results in a leftmost subarray of length 1. This will happen for the rest of array such that the stack call space is $\Theta(n)$

(c)

What we can do is instead of always choosing the leftmost subarray to do the recursive call on, compare the 2 subarrays and choose the smaller subarray. This way the stack space will have to be limited to $\lg n$ as the *smaller* of the two will need to be at most half the lenght of the full array.

Question 2

- the biggest possible depth of a leaf in a decision tree is $n - 1$
    - in the worst case, each element is compared with every other element $n - 1$ before putting it in its right place
    - the tree is imbalanced because only one single node gets seperated from the others
- the smallest possible depth of a leaf is $\lg n$
    - in the best case, the tree is balanced as at every level the number of elements gets divided by two
    - this is because after every comparison, the elements split in half: those less than the element in question and those greater
    - this results in a balanced tree with height $\lg n$

Question 3

No. Counting sort is not good here. The space complexity will be way too big for the ranges $[-29864, 89926187]$. To accomidate the negative value, you would also need to do something like offset the indices in the counting array.

Question 4

```
function RandomizedMedian(A, k) {
    if length(A) == 1 then return A[1]

    p = choose a random pivot from A
    Left, Right = partition A around p

    if k < length(Left) then return RandomizedMedian(Left, K)
    else if k == length(Left) then return p
    else return RandomizedMedian(Right, K - length(Left))
}
```

- this algorithm partitions `A` around some random pivot `p`
- it then checks which partition `k` would fit into
- it terminates either
    - if the `A` is only 1 element
    - or if `k` statistic happens to be the pivot since the pivot is sorted