Name: Trevor Buck
ID: 109081318

CSCI 3104, Algorithms **Profs. Grochow & Layer**
Problem Set 3 **Spring 2019, CU-Boulder**

Hyperlinks for convenience: 1a 1b 1c 2a 2b 2c 3a 3b 3c

1. *(10 pts total) For parts* (1a) *and* (1b), *justify your answers in terms of deterministic QuickSort, and for part* (1c), *refer to Randomized QuickSort. In both cases, refer to the versions of the algorithms given in the lecture notes for Week 3.*

   (a) *(3 points) What is the asymptotic running time of QuickSort when every element of the input A is identical, i.e., for $1 \leq i, j \leq n$, $A[i] = A[j]$? Prove your answer is correct.*

Partition: n swaps of $O(1) = O(n)$

Quicksort ($\leq$) → size of $n-1$

Quicksort ($>$) → empty

Means you would have n layers

$\theta(\text{Quicksort}) = O(\text{layers}) * O(\text{work @ each layer})$

$= O(n) * n = \boxed{O(n^2)}$

Proof:

(IH) IF we are sorting a list of length n, our runtime would be $O(n^2)$

Base case: $n=1$ → Partition = $O(1)$ No more Quicksorts $\geq \boxed{O(1) = O(n^2)}$

Runtime of $O(n+1) = ?$

- 1 more layer to Quicksort $\Rightarrow (n+1)$
- 1 more item to sort $\Rightarrow (n+1)$

$O(\text{Quicksort}(n+1)) = O(\text{layers}) * O(\text{work at each layer})$

$= O(n+1) * (n+1)$

$= \boxed{O(n+1)^2}$

By (IH) we know that if each item in an array is of equal value, it takes $O(n^2)$ time to sort the array.

Name: Trevor Buck
ID: 109081318

CSCI 3104, Algorithms — **Profs. Grochow & Layer**
Problem Set 3 — **Spring 2019, CU-Boulder**

(b) *(3 points) Let the input array A be* $[2, 1, -1, 4, 5, -4, 6, -3, 3, 0]$. *What is the number of times a comparison is made to the element with value* -3 *(note the minus sign)?*

① x=0 2, 1, -1, 4, 5, -4, 6, -3, 3, 0

-1, 1, 2, 4, 5, -4, 6, -3, 3, 0.

-1, -4, 2, 4, 5, 1, 6, -3, 3, 0 ← 1

-1, -4, -3, 4, 5, 1, 6, 2, 3, 0

-1, -4, -3 | 0 | 4, 5, 1, 6, 2, 3

This would sort

② x=-3 -1, -4, -3 ← ②

-4, -1, -3 ← ③

-4 | -3 | -1

③ x=-4 → stop

3 times

(c) *(4 points) How many calls are made to* `random-int` *in (i) the worst case and (ii) the best case? Give your answers in asymptotic notation.*

(i) WORST CASE: All the same value (Question 1)

- We would have (n) layers

So, 'n' times random-int is called

(ii) Best case: Partition splits the array exactly in half every time

- lg(n) layers

So, lg(n) times random-int is called

Name: Trevor Buck
ID: 109081318

CSCI 3104, Algorithms **Profs. Grochow & Layer**
Problem Set 3 **Spring 2019, CU-Boulder**

2. *(20 pts total) Professor Flitwick needs your help. He gives you an array A consisting of n integers $A[1], A[2], \ldots, A[n]$ and asks you to output a two-dimensional $n \times n$ array B in which $B[i,j]$ (for $i < j$) contains the sum of array elements $A[i]$ through $A[j]$, i.e., $B[i,j] = A[i] + A[i+1] + \cdots + A[j]$. (The value of array element $B[i,j]$ is left unspecified whenever $i \geq j$, so it doesn't matter what the output is for these values.)*

Flitwick suggests the following simple algorithm to solve this problem:

```
flitwickSolve(A) {
   for i = 1 to n {
      for j = i+1 to n {
         s = sum(A[i..j])        // look very closely here
         B[i,j] = s
}}}
```

(a) *(5 points) Choose a function f such that $\Omega(f(n))$ serves as a bound on the running time of this algorithm, and prove that this is the case—that is, prove that the worst-case running time is at least $\Omega(f(n))$ on inputs of size n.*

sum: A[i,j-1] + A[i,i] --- f(n)

↑
Has runtime $\Theta(1)$, because it is a constant

Worst case: We have to call our function at every step
↑
we do have to call it at every step so, $n * \Theta(1) = \boxed{\Theta(n)}$

(b) *(5 points) For this same function f you chose in part 2a, show that the running time of the algorithm on any input of size n is also $O(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)*

Because there are no comparisons, we will always call our function exactly once for every step.

- runtime of function = $O(1)$
- n steps (loops) = n

total runtime = $O(1) * n = O(n)$

Name: Trevor Buck

ID: 109081318

CSCI 3104, Algorithms — **Profs. Grochow & Layer**

Problem Set 3 — **Spring 2019, CU-Boulder**

(c) *(10 points) Although Flitwick's algorithm is a natural way to solve the problem—after all, it just iterates through the relevant elements of B, filling in a value for each—it contains some highly unnecessary sources of inefficiency. Give an algorithm that solves this problem in time $O(f(n)/n)$ (asymptotically faster than before) and prove its correctness. (Recall the class conventions on what is required when we ask you to "give an algorithm.")*

```
mysolve(A) {
    for i=1 to n {                    (outer 'for' loop)
        for j=i to n {                (Inner for loop)
            B[i,j] = B[i,j] + A[j]
        }
    }
```

1	2	3	4
1	3	6	10
0	2	5	9
0	0	3	4
0	0	0	4

↑

I'm pretty sure thats exactly what Flitwick had.

??

Name: Trevor Buck
ID: 109081318

CSCI 3104, Algorithms — **Profs. Grochow & Layer**
Problem Set 3 — **Spring 2019, CU-Boulder**

3. *(30 pts total) The Dementors have captured n prisoners, of dubious morals and dubious reliability. The Ministry of Magic needs your help to identify which prisoners can be relied on to tell the truth, and which cannot. To this end, they have developed a spell that can be cast on a pair of prisoners at a time. When the spell is cast, the two prisoners each say whether the other prisoner is truthful or a liar. A truthful prisoner always gives the correct answer—that is, correctly identifies whether the* other *prisoner is truthful or a liar—but anything one of the lying prisoners says cannot be trusted. That is, lying prisoners can lie whenever they want, but they* don't *necessarily always make false statements. Furthermore, the spell has the useful property that if it is cast on the same two prisoners repeatedly, they will always give the same answers they gave initially (hence why a spell is needed).*

You quickly notice that there are four possible outcomes to the spell, when cast on prisoners i and j:

prisoner i says	*prisoner j says*		
"j is a liar"	*"i is a liar"*	$\Longrightarrow$	*at least one is a liar*
"j is a liar"	*"i is truthful"*	$\Longrightarrow$	*at least one is a liar*
"j is truthful"	*"i is a liar"*	$\Longrightarrow$	*at least one is a liar*
"j is truthful"	*"i is truthful"*	$\Longrightarrow$	*both truthful, or both liars*

(a) *(10 points) Prove that if n/2 or more prisoners are liars, the Ministry cannot necessarily determine which prisoners are truthful using* any *strategy based on this kind of pairwise test. Assume a worst-case scenario in which the lying prisoners contain a psychic link that they can use to collectively conspire to fool the Ministry.*

The only way to gain ground is when both prisoners say the other is truthful. You could just separate those cases from the others However if you have more liars than truth tellers, the liars would be able to keep you from distinguishing them. You could get it down to either a group of all liars, or all truthful

(b) *(12 points) Suppose the Ministry knows that strictly more than $n/2$ of the prisoners are truthful, but not which ones. Prove that $\lfloor n/2 \rfloor$ pairwise tests are sufficient to reduce the problem to one of nearly half the size.*

Same as the last problem. If you separate the cases where both people say that the other person is telling the truth, you can eliminate weed out everytime a liar is paired with a truthful person. If you match everyone with everyone else, eventually all of the liars would be paired with truth tellers, and they would be eliminated. In the end, you would have a situation where everyone says that each other is truthful. As long as there are more truthful prisoners to start with, everyone left over will be truthful.

(c) *(8 points) Now, under the same assumptions as part* (3b)*, prove that all of the truthful prisoners can be identified with* $\Theta(n)$ *pairwise tests. Give and solve the recurrence (as always, in* O *or* Θ *notation) that describes the number of tests.*

```
totell the truth (A) {
    int count = 0
    for i = 1 to n/2 {
        if (A[i] = A[i + n/2] = truth) {
            B[count] = A[i]
            B[count + 1] = A[i + n/2]
            count += 2
        }
    }
    totell the truth (B)
```

makes a new array 'B' of only the situation where both say true

for loop = $\Theta(\frac{n}{2})$

layers = $\lg n$ → $\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + \frac{n}{2^{\lg n}}$

$\approx \Theta(n)$