Data Structures - 67109
Exercise 4

Due: 10/04/2019

Question 1

In the Counting Sort algorithm, we assume that the input is an array $A[1...n]$ (and thus $A.length = n$), and an array $B[1..n]$ that eventually holds the sorted output.

```
counting_sort(A,B,k):
    let C[0...k] be a new array full of zeros
    for j = 1 to A.size
        C[A[j]] += 1
    for i = 1 to k
        C[i] += C[i-1]
    for j = A.size downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] -= 1
    return B
```

https://www.youtube.com/watch?v=7zuGmKfUt7s - is a youtube video explaining Counting sort. Note that it is different from the pseudo-code version above.

1. Prove that (our) Counting Sort is stable.
2. Suppose that we were to rewrite the last for loop header (for j = A.size downto 1) of the Counting-Sort pseudo code as: for j=1 to A.size (This is the version shown in the video). Show that the algorithm still works properly. Is the modified algorithm stable?
3. Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $\{a, \ldots, b\}$ in $O(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time (hint: re-read the Counting Sort algorithm from the recitation). Prove the correctness of your algorithm and the asymptotic bounds you claim it has.

Question 2

1. Recall the definition from the lecture of $\mathbb{E}[T(n)]$:

$$R(n) = \sum_{m=1}^{n} \frac{1}{n} \left(R(m-1) + R(n-m)\right) + \Theta(n) = \ldots = \frac{2}{n} \sum_{m=1}^{n-1} R(m) + \Theta(n)$$

   Looking for an upper bound for $R(n)$, we defined $U(n) = cn + \frac{2}{n} \sum_{m=1}^{n-1} U(m)$ and we have shown it's bounded by $O\left(n \log(n)\right)$ and we used the harmonic series to reach that bound
   Prove by induction that $U(n)$ is indeed an upper bound for $R(n)$.

2. Prove that $lim_{n\to\infty} \sum_{i=1}^{n} \frac{(-1)^{i+1}}{i} = ln\, 2$, you may assume the limit exists. (Hint: You might want to use the fact that $\lim_{n\to\infty} \sum_{i=1}^{n} \frac{1}{i} - ln\left(n\right) = \gamma < \infty$ and $\sum_{i=1}^{2n} \frac{(-1)^{i+1}}{i} = \sum_{i=1}^{2n} \frac{1}{i} - 2\sum_{i=1}^{n} \frac{1}{2i}$)

Question 3

Let $Godiva[1, \ldots, n]$ be a chocolate bar with n units.

1. Let us choose some index $k \in \{0, \ldots, n\}$ uniformly at random, and break the bar at that point to share with a friend (i.e. if k was chosen, we are left with two pieces $Godiva[1, \ldots, k], Godiva[k+1, \ldots, n]$) (if $k == 0$ then the left side is an empty piece and the same for the right side if $k == n$). We love chocolate, so we always save the larger piece for ourselves. What is the expected size of the piece we will get? (You may approximate the value when dealing with non-integers)

2. Recall the proof of the expected running time of the **random** Quick-Select. During the proof we noted that although the expected size of the subarray we will call the method on is $\frac{3}{4}n$, we cannot use it as the parameter in the reccurrence relation for Quick-Select's running time. Explain why that is the case (no need to be formal)

3. Give an example for a function $f : \{1, \ldots, n\} \to [0, n]$ and a distribution (i.e. probability function) over $\Omega = \{1, \ldots, n\}$ such that $\mathbb{E}\left[f\left(i\right)\right] > max\left\{f\left(\lceil \mathbb{E}\left[i\right] \rceil\right), f\left(\lfloor \mathbb{E}\left[i\right] \rfloor\right)\right\}$ where i is a random variable $i(\omega) = \omega$.

Question 4

Recall the random Quick-Select algorithm seen in recitation. It's expected running time is $O\left(n\right)$. Can we find a **deterministic** algorithm that runs in linear time? The answer is yes. We know the best choice for a pivot is the median, but finding the median is exactly finding the $\frac{n}{2}^{th}$ largest number. We then settle

for finding some element in our array such that there are at least $\frac{3}{10}n$ elements smaller than it, and $\frac{3}{10}n$ elements greater than it. This cell will be a sort of approximation for the median.

The deterministic version of the PARTITION function is thus:

1. Group the array into $n/5$ groups of size 5, and find the median of each group (for simplicity assume $n = 5^k$)
2. Find the median of the medians found in 1 using quickselect (recursively). Denote it p.
3. Swap the array into the appropriate subarrays (all elements greater than $arr[p]$ to the right of it, and all elements smaller than it to its left)
4. Return p's index in the rearranged array.

Answer the following questions:

1. Prove that p indeed has at least $\frac{3}{10}n$ elements smaller than it, and $\frac{3}{10}n$ elements greater than it. (Hint: let $g = n/5$ be the number of subarrays. How many of them **surely** contain elements smaller-or-equal than p? Which elements in those subarrays are **surely** smaller-or-equal than p? Count these elements. Same for the other direction)
2. Try to think of an upper bound for the worst case running time of the above steps (no need to write an answer)
3. Write a recurrence relation to bound the running time of Quick-Select with the new PARTITION function.
4. Prove that the recurrence relation from 3 is $O(n)$.
5. This algorithm has been created in 1972 and has caused a lot of surprise in the Computer Science community (where the common belief was that there does not exist a linear-time deterministic algorithm for the problem). Why was this result surprising?