Honesty Pledge

Illinois Institute of Technology
Department of Computer Science

Honesty Pledge

CS 430 Introduction to Algorithms
Spring Semester, 2016

Fill out the information below, sign this sheet, and submit it with the first homework assignment. No homework will be accepted until the signed pledge is submitted.

I promise, *on penalty of failure of CS 430*, not to collaborate with anyone, not to seek or accept any outside help, and not to give any help to others on the homework problems in CS 430.

All work I submit will be mine and mine alone.

I understand that all resources in print or on the web, aside from the text and class notes, used in solving the homework problems *must be explicitly cited.*

Sairam Kannan	K. Sairam	A20355970	01/20/2016
Name (printed)	Signature	Student ID	Date

Homework Assignment 2

1.

Homework Assignment 2

1. (a) Let $\Pi = (\pi_1, \pi_2, \dots \pi_n)$ be a random permutation of $\{1, 2 \dots n\}$.

To find: Expected Value of $\frac{1}{n}\sum_{i=1}^{n} |\pi_i - i|$

$$\cancel{\frac{1}{n^2}\sum_{\pi_t=t}^{n} |\pi_i - i| = \frac{1}{n+1}}$$

The expected value of $\frac{1}{n}\sum_{i=1}^{n} |\pi_i - i|$ is $\frac{1}{n}\sum_{\pi_t=1}^{n} |\pi_i - i|$ averaging around i. [as per the assumption made in the question]

$$= \frac{1}{n}\sum_{\pi_t=1}^{n} |\pi_i - i| = \frac{1}{n}\left[\sum_{\pi_t=1}^{i}(i-\pi_i) + \sum_{\pi_t=i+1}^{n}(\pi_i - i)\right]$$

[Splitting the intervals]

$$= \frac{1}{n}\left(\sum_{\pi_t=1}^{i-1}\pi_i + \sum_{\pi_t=1}^{n-i}\pi_i\right)$$

$$= \frac{1}{2n}\left[i^2 - i + (n-i)(n-i+1)\right]$$

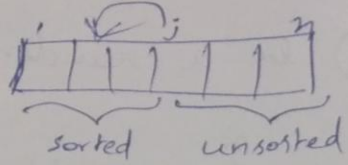
$$= \frac{1}{2n^2}\sum_{i=1}^{n}\left[2i^2 - 2(n+1)i + n^2 + n\right] \Rightarrow \left(\text{Average around } i\right)$$

$$= \frac{1}{3n}(n^2 - 1) = \frac{n}{3} - \frac{1}{3n}$$

(b) The summation from $i = 1$ to n, gives the total steps to sort the array. The ~~[illegible]~~ Absolute value of $|\pi_i - i|$ describes about the total distance of an element, from its sorted position. Therefore the average distance of an

1 continuation and 2nd Problem.

Insertion sort (data movement)



j^{th} element $\rightarrow$ $(j-1)$ element.

item moved during sorting will be the summation of all these elements divided by the total number of elements

(c) If its an Insertion kind of sorting algorithm which performs adjacent interchanges, the total time will be the product of average distance travelled by the element and the total number of elements. $\therefore\ n \times \frac{1}{3n}(n^2-1) = \frac{1}{3}(n^2-1)$ $= \Theta(n^2)$. Thus sorting Algorithm using adjacent interchanges takes time of the order of $\Theta(n^2)$; where n is number of elements.

(2) Problem 6.4-3 (Page 160)

If the Array A of length n is already sorted in the descending order, it is the best case schenario, since the largest element is always on the top. Thus to check if the Array is heap, the BUILD_MAX_HEAP should run through

2nd problem continuation and 3rd Problem

most of the elements present in the Array. Therefore it takes $\Theta(n)$, even though the Array is max_heap. For every MAX_HEAPIFY Call, one of the smallest values in the heap is kept at the top of the heap. It takes $\Theta(\log n)$ time, to move the smallest element to the bottom of the heap. [since heap will be of ~~size~~ depth $\log n$]

$\therefore$ ~~worst~~ Best case running time is $\Theta(n \log n)$

If the Array is sorted in the increasing order, BUILD_MAX_HEAP will take $\Theta(n)$ to build the heap. (But the Comparison will be more as compared to the previous case, when the array is in descending order). However for each call to the function MAX_HEAPIFY, it takes the worst case time of the order of Lower bound of $\lg n$. $(\Omega \lg n)$. So the Algorithm takes $\Omega(n \lg n)$.

(3) Problem 7-4 Page (188).

(a). In a Tail Recursive sort (of array sorted in ascending order) the partition function, will partition the elements in such a way that the first $(n-1)$ elements falls in the left sub list and the left sub list is recursively sorted by calling the Tail Recursive

3rd Problem continuation

Quicksort function when each recursive call is called, the left sublist length is decreased by one. Thus we have $\Theta(n)$ recursive calls. As a result the stack depth will be $\Theta(n)$

(C) The recursive relation for the Tail Recursive Quicksort (TRQ) is.

$$f(n) = 1 + \frac{1}{n} \sum_{i=1}^{n-1} S(i).$$

Let the initial values be $t_0, t_1, \dots t_n$ for $n_{i=0}$

we can rewrite as.

$$t_n = a + \frac{1}{n} \sum_{i=1}^{n-1} t_i \quad (n > n_0) \quad — ①$$

$$(n-1)\, t_{n-1} = a(n-1) + \sum_{i=1}^{n-2} t_i \quad \text{(Sub } (n-1) \text{ for } n) \quad ②$$

Solving ① + ②, we get.

$$nt_n - n t_{n-1} + t_{n-1} = \cancel{an} - \cancel{an} + 1 + t_{n-1}$$

Since, $nt_n - nt_{n-1} = a \Rightarrow t_n - t_{n-1} = a/n$

$$\therefore \sum_{i=1}^{n} t_n - t_{n-1} = \sum_{i}^{n} a/i$$

$$\therefore t_n = a \lg n = \Theta(\lg n)$$

$\therefore$ Total Running time will be $\Theta(\lg n) * \Theta(n) = O(n \lg n)$

3rd Problem continuation

(a). The Traditional Quicksort and Tail Recursive Quicksort, does the same partitioning, expect with the difference being traditional quicksort Calles itself again with arguments $A, q+1, r$ while (TRQ) Calls itself after setting $p \leftarrow q+1$.

Both the sorting algorithms perform the same operation across the entire array, as in both Cases the third argument (A, r) have the same values and p has the old value of $(q+1)$.

4th Problem continuation and 5th Problem

4. Problem 8.3(a). (Page 206)

Radix sort can be used to solve this. If the numbers have at most 't' digits. From $i = 1$ to t, sorting takes place according to i^{th} ~~characters~~ digit of each number

Let d_i, be the number of ~~digits~~, integers that have their i_{th} digit. This stable sorting algorithm, takes time which is probably less than $\sum_{i=1}^{t} i \cdot c(d_i + D)$ time [c + D is a constant]

$$\sum_{i=1}^{t} i \cdot c(d_i + D) = c\sum_{i=1}^{t} i d_i + D\sum_{i=1}^{t} i \le (c+D)n$$

$$= O(n)$$

5. Randomized-Select (A, p, r, i)

```
if p == r
    return A[p]
q = Randomized_partition (A, p, r)
k = q - p + 1
if i == k
    return A[q]
elseif i < k
    return Randomized_Select (A, p, q-1, i)
```

5th Problem continuation.

else return Randomized_Select $(A, q+1, r, i-k)$.

(a) In the above Algorithm, on line 8 if $q-1$ is replaced by q, the corrupted code will work sometimes. ~~If~~ The running of the code depends on the selection of pivot element. If the subarray $A[p \ldots q]$ created out of partition is same as the original array $A[p \ldots r]$, the code will not terminate, as it keeps looping back on the same function RANDOMIZED_PARTITION.

(b) The worst-case running time will be infinite, since the code will not terminate.

(c) The best_case behaviour, when the i^{th} smallest element is selected as the pivot element in the first partition. Since the function is recursive among all the elements present in the array, the order of the algorithm is $O(n)$, where n is the total number of elements present in the array.

5th Problem continuation.

Average case Analysis.

(d) $T(n) \leq \sum_{k=1}^{n} X_k \cdot T(\max(k, n-k)) + O(n)$ ①

~~observe~~ $\max(k, n-k) = \begin{cases} k & \text{if } k \geq \lceil n/2 \rceil \\ n-k & \text{if } k < \lceil n/2 \rceil \end{cases}$ ②

Sub ② in ①, we have.

$$E[T(n)](1 - 1/n) \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} E[T(k)] + O(n)$$

$$E[T(n)](1 - 1/n) \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} ck + an$$

$$= c\left(\frac{3n}{4} - \frac{1}{2}\right) + an.$$

$$\therefore E[T(n)] \leq cn$$

which proves that $E[T(n)] = O(n)$

(e). The Average Case running time is $O(n)$ which is mentioned in the previous step. Since the pivot is selected randomly across n elements in the Array, the behavior remains linear with respect to the growth of data elements in the Array. If the same element appears repeatedly, the probability of selecting same element will be more by a constant, but overall running time will be $O(n)$.