Homework 5 Solutions

1. Counting Sort, Radix Sort, Bucket Sort (30 points)

(1) COUNTING - SORT : $A = \langle 6, 0, 2, 6, 0, 8 \rangle$

A

1	2	3	4	5	6
6	0	2	6	0	8

C

0	1	2	3	4	5	6	7	8
2	0	1	0	0	0	2	0	1

C

0	1	2	3	4	5	6	7	8
2	2	3	3	3	3	5	5	6

B

1	2	3	4	5	6
					8
	0				8
	0			6	8
	0	2		6	8
0	0	2		6	8
0	0	2	6	6	8

C

0	1	2	3	4	5	6	7	8
2	2	3	3	3	3	5	5	5
1	2	3	3	3	3	5	5	5
1	2	3	3	3	3	4	5	5
1	2	2	3	3	3	4	5	5
0	2	2	3	3	3	4	5	5
0	2	2	3	3	3	3	5	5

Answer B

0	0	2	6	6	8

RADIX - SORT :

Given Input:

PAT
CAT
CART
FAT
FIX

→

PAT
CAT
CART
FAT
FIX

→

PAT
CAT
FAT
FIX
CART

→

CART
CAT
FAT
FIX
PAT

→

CAT
FAT
FIX
PAT
CART

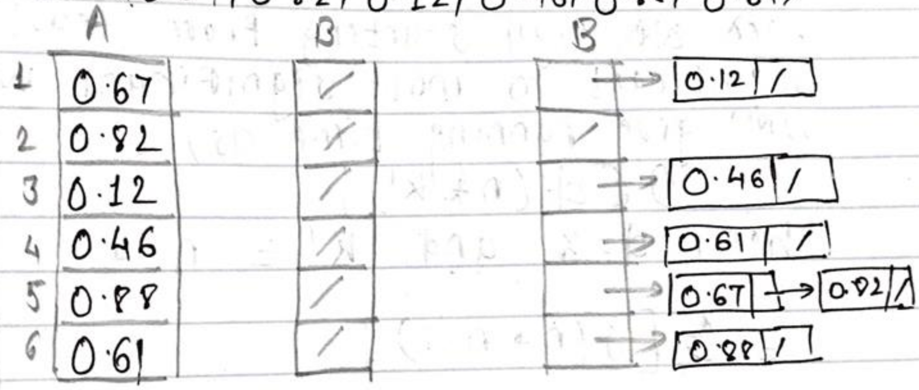
=

Output

CAT
FAT
FIX
PAT
CART

Q.3) BUCKET-SORT

$A = \langle 0.67, 0.82, 0.12, 0.46, 0.88, 0.61 \rangle$



$\text{floor}(0.67 \times 6) = \text{floor}(4.02) = 4$

$\text{floor}(0.82 \times 6) = \text{floor}(4.92) = 4$

$\text{floor}(0.12 \times 6) = \text{floor}(0.72) = 0$

$\text{floor}(0.46 \times 6) = \text{floor}(2.76) = 2$

$\text{floor}(0.88 \times 6) = \text{floor}(5.28) = 5$

$\text{floor}(0.61 \times 6) = \text{floor}(3.66) = 3$

2. Counting Sort, Radix Sort

Show how to sort n integers in the range 0 to n^3 - 1 in O(n) time

using radix sort we have 3 digits in base n So we call counting Sort three times

2) *What is the running time if we use Counting Sort? Justify your answer.*

$O(n^3)$ because the range of the input is $n^3 - 1$

$O(n+K)$ K = range of input

$= O(n + (n^3 - 1))$

$= O(n^3)$

3. Sorting

Sorting – Explain why the worst-case running time for bucket sort is $\theta(n^2)$. What simple change to the algorithm preserves its linear average-case running time and makes its worst-case running time $O(n \lg n)$?

The worst-case running time for bucket sort occurs when a single bucket contains all n elements of the original array. After placing the elements into their appropriate bucket, Insertion-Sort is called to sort them in the bucket which has a worst-case running time of $O(n^2)$. The dominating cost of Bucket-Sort is in sorting each bucket so that can be easily fixed by replacing Insertion-Sort with a different sorting algorithm that has a better worst-case running time. For instance, merge sort has a worst-case running time of $O(n \lg n)$ and can be called to sort each bucket to give Bucket-Sort a worst-case running time also of $O(n \lg n)$.

4. Exercise 9.3-3

If we rewrite `PARTITION` to use the same approach as `SELECT`, it will perform in $\mathcal{O}(n)$ time, but the smallest partition will be at least one-fourth of the input (for large enough n, as illustrated in exercise 9.3.2). This will yield a worst-case recurrence of:

$$T(n) = T(n/4) + T(3n/4) + \mathcal{O}(n)$$

As of exercise 4.4.9, we know that this is $\Theta(n \lg n)$.

And that's how we can prevent quicksort from getting quadratic in the worst case, although this approach probably has a constant that is too large for practical purposes.

Another approach would be to find the median in linear time (with `SELECT`) and partition around it. That will always give an even split.

5. Exercise 9.3-5

We find the median in linear time partition the array around it (again, in linear time). If the median index (always $\lceil n/2 \rceil$) equals n we return the median. Otherwise, we recurse either in the lower or upper part of the array, adjusting n accordingly.

This yields the following recurrence:

$$T(n) = T(n/2) + \mathcal{O}(n)$$

Applying the master method, we get an upper bound of $\mathcal{O}(n)$.

6. Exercise 9.3-6

1. If $k = 1$ we return an empty list.
2. If k is even, we find the median, partition around it, solve two similar subproblems of size $\lfloor n/2 \rfloor$ and return their solutions plus the median.
3. If k is odd, we find the $\lfloor k/2 \rfloor$ and $\lceil k/2 \rceil$ boundaries and the we reduce to two subproblems, each with size less than $n/2$. The worst case recurrence is:

$$T(n,k) = 2T(\lfloor n/2 \rfloor, k/2) + O(n)$$

Which is the desired bound $\mathcal{O}(n \lg k)$.

This works easily when the number of elements is $ak + k - 1$ for a positive integer a. When they are a different number, some care with rounding needs to be taken in order to avoid creating two segments that differ by more than 1.

7. Problem 9-1

Sorting

We can sort with any of the $n \lg n$ algorithms, that is, merge sort or heap sort and then just take the first i elements linearly.

This will take $n \lg n + i$ time.

Max-priority queue

We can build the heap linearly and then take each of the largest i elements in logarithmic time.

This takes $n + i \lg n$.

Partition and sort

Let's assume we use the `SELECT` algorithm from the chapter. We can find the ith order statistic and partition around it in n time and then we need to do a sort in $i \lg i$.

This takes $n + i \lg i$