Sabancı University
Faculty of Engineering and Natural Sciences

CS301 – Algorithms

Homework 2

Due: March 12, 2024 @ 23.55
(upload to SUCourse)

Kenat Özyer
0003073
S. Özyer

PLEASE NOTE:

- Provide only the requested information and nothing more. Unreadable, unintelligible, and irrelevant answers will not be considered.
- Submit only a PDF file. (-20 pts penalty for any other format)
- Not every question of this homework will be graded. We will announce the question(s) that will be graded after the submission.
- You can collaborate with your `TA/INSTRUCTOR ONLY` and discuss the solutions of the problems. However, you have to write down the solutions on your own.
- Plagiarism will not be tolerated.

Late Submission Policy:

- Your homework grade will be decided by multiplying what you normally get from your answers by a "submission time factor (STF)".
- If you submit on time (i.e. before the deadline), your STF is 1. So, you don't lose anything.
- If you submit late, you will lose 0.01 of your STF for every 5 mins of delay.
- We will not accept any homework later than 500 mins after the deadline.
- SUCourse's timestamp will be used for STF computation.
- If you submit multiple times, the last submission time will be used.

Question 1

(a) What is the form of the input array that triggers the worst case of the insertion sort?

The one that is sorted in the reverse order.

(b) What is the complexity of this worst–case behavior in Θ notation?

The worst-case behavior is $\Theta(n^2)$

(c) Explain how this particular form of the array results in this complexity.

If the array is sorted in reverse, the following happens:

Every element gets compared with every other element. For the insertion step the newly added element gets propagated to the back of the array. This happens for every single step thus creating n^2 comparisons to be made.

Question 2

(a) What is the form of the input array that triggers the best case of the insertion sort?

The best case scenario is when the array is already sorted.

(b) What is the complexity of this best–case behavior in Θ notation?

The complexity of the best case behavior is $\Theta(N)$

(c) Explain how this particular form of the array results in this complexity.

This is due to the following reason;
When the algorithm iterates over the elements, there exists no need to back-propagate items back to the already-sorted array. This is due to the fact that the upcoming element will always be bigger than the last element of the already-sorted part, which is good because only N comparisons will be made.

Question 3

Which of the following sorting algorithms are stable: insertion sort, merge sort, heapsort, and quicksort? Give a simple scheme that makes any comparison sort stable. How much additional time and space does your scheme entail?

Among the given sorting algorithms, only merge sort is a stable sorting algorithm by default.

Every single element should have their index.
In the event of a tie, element that is of biggest index is put as the bigger element.

For time complexity, only a overhead for the tie-breaker situations will occur, in the case of identical elements.

For space complexity, an 64-bit integer to store every element. A space complexity overhead of $O(n)$ will be enough

Question 4

(a) Given n d-digit numbers in which each digit can take on up to k possible values, RADIX-SORT correctly sorts these numbers in $O(d(n+k))$ time if the intermediate stable sorting algorithm is Counting Sort.

$O(d(n+k))$ where k is the range of counting sort

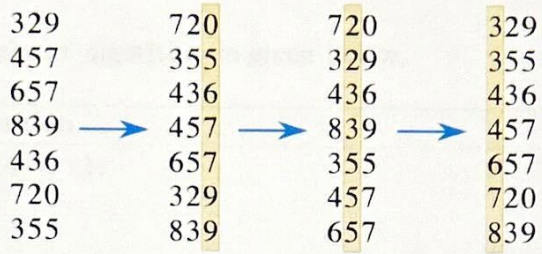

Figure 1: The operation of radix sort on seven 3-digit numbers. The leftmost column is the input. The remaining columns show the numbers after successive sorts on increasingly significant digit positions. Tan shading indicates the digit position sorted on to produce each list from the previous one.

(b) Using Figure 1 as a model, illustrate the operation of RADIX-SORT on the following list of English words: COW, DOG,SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX

1st Iteration

SEA, TEA, MOB, TAB, DOG, RUG, DIG, BIG, BAR, EAR, TAR, COW, ROW, NOW, BOX, FOX

2nd Iteration

TAB, BAR, EAR, TAR, SEA, TEA, DIG, BIG, DOG, COW, ROW, NOW, BOX, FOX, RUG

3rd Iteration

BAR, BIG, BOX, COW, DIG, DOG, EAR, FOX, NOW, ROW, RUG, SEA, TAB, TAR, TEA

Question 5

The pseudo-code for Quicksort algorithm is given below.

Algorithm 1 Quicksort algorithm

```
Function Quicksort(array A, l, r):
    if r − l + 1 ≤ 1 then
        return
    end
    p ← ChoosePivot(A, l, r)
    Partition(A, p, l, r)
    Quicksort(A, l, p − 1)
    Quicksort(A, p + 1, r)

Function Partition(A, p, l, r):
    i ← l + 1
    for j ← l + 1 to r do
        if A[j] ≤ p then
            swap A[i] with A[j]
            i ← i + 1
        end
    end
    swap A[i − 1] with p
    return i − 1
Function ChoosePivot(A, l, r):
    return A[⌊(l + r)/2⌋]
```

(a) Write down the recurrence for the running time for the case where the algorithm chooses the median as the pivot at each iteration.

In this case, the time complexity is $O(n \log n)$

(b) Calculate a tight bound for this recurrence using the Master Theorem.

Quicksort algorithm can be defined by this recurrence:

$$T(n) = 2T(\tfrac{n}{2}) + \Theta(1) + \Theta(n)$$

Recurrence creates two sub-problems of half the size of the original problem by the fact that the pivot is the median. Finding the median is a $\Theta(1)$ indexing operation. Partitioning is a $\Theta(n)$ operation.

$a = 2$, $b = 2$, $f(n) = \Theta(1) + \Theta(n)$ which is asymptotically positive.

$\log_b a = \log_2 2 = 1 \Rightarrow f(n) \neq \Theta(n^{1-\varepsilon})$ for some $\varepsilon > 0$, case I fails.

$\Rightarrow f(n) = \Theta(n^{\log_2 2})$ holds, therefore, case II holds.

Hence, the complexity of the Quicksort algorithm is $\Theta(n \log n)$ for when the pivot is median.

(c) [5 points] Write down the recurrence for the running time for the case where the algorithm chooses the smallest element in the array as the pivot at each iteration.

The recurrence relation of this case is

$T(n) = T(n-1) + \Theta(n)$ since finding smallest element is $\Theta(n)$ and the rest will go without the pivot, to the one side

(d) [5 points] Calculate a tight bound for this recurrence using the iteration method.

Given the recurrence relation:

$$T(n) = T(n-1) + \Theta(n)$$

Applying the iteration method above:

$$T(n) = T(n-1) + \Theta(n)$$
$$\Rightarrow T(n-2) + \Theta(n) + \Theta(n)$$
$$\Rightarrow T(n-3) + \Theta(n) + \Theta(n) + \Theta(n)$$
$$\cdots$$
$$\underbrace{T(1)}_{\Theta(1)} + \underbrace{\Theta(n) + \cdots + \Theta(n)}_{n\text{-times}}$$

Hence, the completely tight bound of the recurrence is

$$\Theta(n^2)$$