CS202 Fall 2021–2022 Homework 1

# Algorithm Efficiency and Sorting

Can Avşar 21902111

# Question 1

## Part A: Finding Asymptomatic Running Times

### 1st Recurrence Equation

$T(n) = 3T(n/3) + n$
$T(n/3) = 3T(n/9) + n/3$
$T(n) = 3[3T(n/9) + n/3 ] + n = 9T(n/9) + 2n$
$T(n/9) = 3T(n/27) + n/9$
$T(n) = 3\{3[3T(n/27) + n/9] + n/3\} + n = 27T(n/27) + 3n$

$T(n) = 3T(n/3) + n$
$T(n/3) = 3T(n/9) + n/3$
$T(n) = 3[3T(n/9) + n/3] + n$
$T(n) = 9T(n/9) + 2n$

**General Form**

$3^k * T(n/3^k) + kn$

**Write $n/3^k$ should be 1. So $n = 3^k$ and $k = \log_3 n$**

$T(n) = 3^{\log_3 n} * T(n/3^{\log_3 n}) + \log_3 n * n$

$T(n) = n*T(1) + \log_3 n * n$

$T(n) = \Theta(n * \log n)$

### 4th Recurrence Equation

$T(n) = 3T(n/2) + 1$
$T(n/2) = 3T(n/4) + 1$
$T(n) = 3[3T(n/4) + 1] + 1 = 9T(n/4) + 4$
$T(n/4) = 3T(n/8) + 1$
$T(n) = 9[3T(n/8) + 1] + 4 = 27T(n/8) + 13$

**General Form 1+3+9**

$3^k * T(n/2^k) + (3^{k+1} - 1) / 2$

**Write $n/2^k$ should be 1. So $n = 2^k$ and $k = \log_2 n$**

$T(n) = 3^{\log_2 n} * T(n/2^{\log_2 n}) + (3^{\log 2n+1} - 1) / 2$

$T(n) = 3^{\log_2 n} * T(1) + (3^{\log_2 n+1} - 1) / 2$

$T(n) = n^{\log_2 3} * T(1) + (n^{\log_2 3+1} - 1) / 2$

$T(n) = \Theta(n^{\log n})$

# Part B: Tracing Sorting Algorithms

## Bubble Sort

The sorted area is shown underlined.

| Steps | Array |
|---|---|
| Take the first element of unsorted part and if the next item is smaller, swap them | 5, 6, 8, 4, 10, 2, 9, 1, 3, 7 |
| The next element is bigger | 5, 6, 8, 4, 10, 2, 9, 1, 3, 7 |
| The next element is bigger | 5, 6, 8, 4, 10, 2, 9, 1, 3, 7 |
| If the next item is smaller, swap them | 5, 6, 4, 8, 10, 2, 9, 1, 3, 7 |
| The next element is bigger | 5, 6, 4, 8, 10, 2, 9, 1, 3, 7 |
| If the next item is smaller, swap them | 5, 6, 4, 8, 2, 9, 1, 3, 7, 10 |
| Take the first element of unsorted part and if the next item is smaller, swap them | 5, 6, 4, 8, 2, 9, 1, 3, 7, 10 |
| The next element is bigger | 5, 6, 4, 8, 2, 9, 1, 3, 7, 10 |
| If the next item is smaller, swap them | 5, 4, 6, 8, 2, 9, 1, 3, 7, 10 |
| The next element is bigger | 5, 4, 6, 8, 2, 9, 1, 3, 7, 10 |
| If the next item is smaller, swap them | 5, 4, 6, 2, 8, 9, 1, 3, 7, 10 |
| The next element is bigger | 5, 4, 6, 2, 8, 9, 1, 3, 7, 10 |
| If the next item is smaller, swap them | 5, 4, 6, 2, 8, 1, 3, 7, 9, 10 |
| Take the first element of unsorted part and if the next item is smaller, swap them | 5, 4, 6, 2, 8, 1, 3, 7, 9, 10 |
| If the next item is smaller, swap them | 4, 5, 6, 2, 8, 1, 3, 7, 9, 10 |
| The next element is bigger | 4, 5, 6, 2, 8, 1, 3, 7, 9, 10 |
| If the next item is smaller, swap them | 4, 5, 2, 6, 8, 1, 3, 7, 9, 10 |
| The next element is bigger | 4, 5, 2, 6, 8, 1, 3, 7, 9, 10 |
| If the next item is smaller, swap them | 4, 5, 2, 6, 1, 3, 7, 8, 9, 10 |
| Take the first element of unsorted part and if the next item is smaller, swap them | 4, 5, 2, 6, 1, 3, 7, 8, 9, 10 |
| The next element is bigger | 4, 5, 2, 6, 1, 3, 7, 8, 9, 10 |
| If the next item is smaller, swap them | 4, 2, 5, 6, 1, 3, 7, 8, 9, 10 |
| The next element is bigger | 4, 2, 5, 6, 1, 3, 7, 8, 9, 10 |
| If the next item is smaller, swap them | 4, 2, 5, 1, 3, 6, 7, 8, 9, 10 |
| The next element is bigger | 4, 2, 5, 1, 3, 6, 7, 8, 9, 10 |

| | |
|---|---|
| Take the first element of unsorted part and if the next item is smaller, swap them | 4, 2, 5, 1, 3, 6, 7, 8, 9, 10 |
| If the next item is smaller, swap them | 2, 4, 5, 1, 3, 6, 7, 8, 9, 10 |
| The next element is bigger | 2, 4, 5, 1, 3, 6, 7, 8, 9, 10 |
| If the next item is smaller, swap them | 2, 4, 1, 3, 5, 6, 7, 8, 9, 10 |
| The next element is bigger | 2, 4, 1, 3, 5, 6, 7, 8, 9, 10 |
| Take the first element of unsorted part and if the next item is smaller, swap them | 2, 4, 1, 3, 5, 6, 7, 8, 9, 10 |
| The next element is bigger | 2, 4, 1, 3, 5, 6, 7, 8, 9, 10 |
| If the next item is smaller, swap them | 2, 1, 3, 4, 5, 6, 7, 8, 9, 10 |
| The next element is bigger | 2, 1, 3, 4, 5, 6, 7, 8, 9, 10 |
| Take the first element of unsorted part and if the next item is smaller, swap them | 2, 1, 3, 4, 5, 6, 7, 8, 9, 10 |
| If the next item is smaller, swap them | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 |
| It iterates through the end and all items are larger than left one | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 |
| (Array is sorted) | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 |

## Selection Sort

The unsorted area is shown underlined.

| Steps | Array |
|---|---|
| Find the biggest element in the unsorted area (10) | 5, 6, 8, 4, 10, 2, 9, 1, 3, 7 |
| Swap with the last element in the unsorted area (10 <—> 7) | 5, 6, 8, 4, 7, 2, 9, 1, 3, 10 |
| Find the biggest element in the unsorted area (9) | 5, 6, 8, 4, 7, 2, 9, 1, 3, 10 |
| Swap with the last element in the unsorted area (9 <—> 3) | 5, 6, 8, 4, 7, 2, 3, 1, 9, 10 |
| Find the biggest element in the unsorted area (8) | 5, 6, 8, 4, 7, 2, 3, 1, 9, 10 |
| Swap with the last element in the unsorted area (8 <—> 1) | 5, 6, 1, 4, 7, 2, 3, 8, 9, 10 |

| | |
|---|---|
| Find the biggest element in the unsorted area (7) | 5, 6, 1, 4, 7, 2, 3, 8, 9, 10 |
| Swap with the last element in the unsorted area (7 <—> 3) | 5, 6, 1, 4, 3, 2, 7, 8, 9, 10 |
| Find the biggest element in the unsorted area (6) | 5, 6, 1, 4, 3, 2, 7, 8, 9, 10 |
| Swap with the last element in the unsorted area (6 <—> 2) | 5, 2, 1, 4, 3, 6, 7, 8, 9, 10 |
| Find the biggest element in the unsorted area (5) | 5, 2, 1, 4, 3, 6, 7, 8, 9, 10 |
| Swap with the last element in the unsorted area (5 <—> 3) | 3, 2, 1, 4, 5, 6, 7, 8, 9, 10 |
| Find the biggest element in the unsorted area (4) | 3, 2, 1, 4, 5, 6, 7, 8, 9, 10 |
| Swap with the last element in the unsorted area (4 <—> 4) | 3, 2, 1, 4, 5, 6, 7, 8, 9, 10 |
| Find the biggest element in the unsorted area (3) | 3, 2, 1, 4, 5, 6, 7, 8, 9, 10 |
| Swap with the last element in the unsorted area (3 <—> 1) | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 |
| (Array is sorted) | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 |

## Part C: Quick Sort Worst Case Recurrence Relation

We select the first element and we do it again on the array which has a size n-1.

T(n) = n + T(n-1)

T(n-1) = (n-1) + T(n-2)

T(n-2) = (n-2) + T(n-3)

.

.

.

T(n) = n + (n - 1) + . . . + 2 + 1 + 0

= (n^2) / 2

The time complexity of worst case of quicksort is O(n^2)

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

$T(n-1) = n-1 + T(n-2)$

$T(n) = n + n-1 + T(n-2)$

. . .

$n + n-1 + n-2 + \cdots 0 = \frac{n^2}{2}$

$\frac{n \cdot n-1}{2} = \frac{n^2 - n}{2}$

$(\frac{n}{2})^2 = \frac{n^2}{4}$ ?

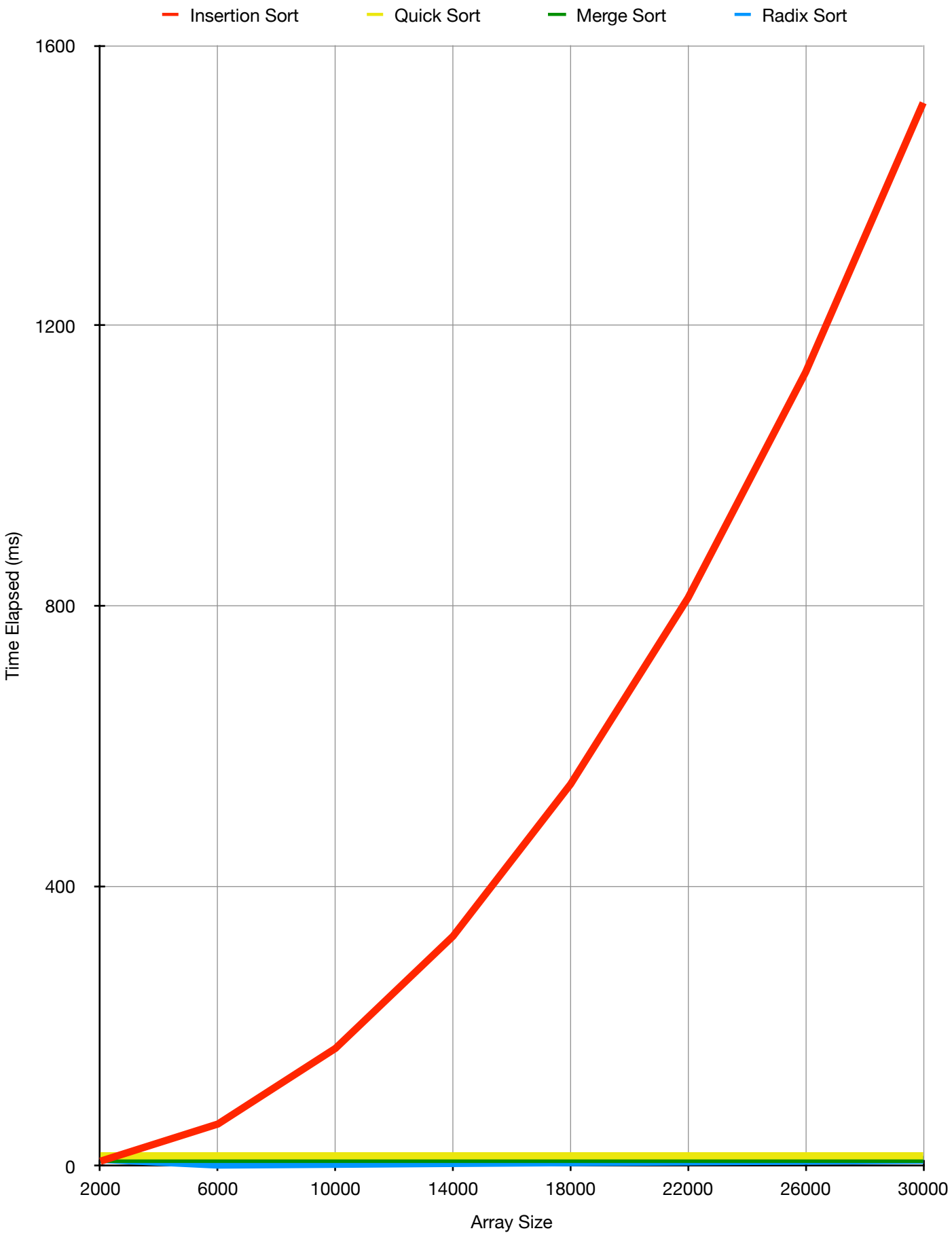
# Question 2

## Table for Sorting Algorithms

```
------------------------------------------------------------
Part A: Time Analysis of Insertion Sort
Array Size      Time Elapsed      compCount       moveCount
   2000         7.338 ms         1003391         1005390
  6000         60.467 ms         8860258         8866257
  10000         168.481 ms         24864314         24874313
  14000         328.599 ms         48385690         48399689
  18000         545.497 ms         80230642         80248641
  22000         811.728 ms         119456731         119478730
  26000         1133.75 ms         167932482         167958481
  30000         1518.38 ms         224421139         224451138
------------------------------------------------------------
Part B: Time Analysis of Merge Sort
Array Size      Time Elapsed      compCount      moveCount
   2000         1.013 ms         19437         43904
  6000         2.196 ms         67865         151616
  10000         3.478 ms         120428         267232
  14000         4.767 ms         175337         387232
  18000         6.141 ms         231953         510464
  22000         7.517 ms         290194         638464
  26000         8.933 ms         348844         766464
  30000         10.34 ms         408519         894464
------------------------------------------------------------
Part C: Time Analysis of Quick Sort
Array Size      Time Elapsed      compCount      moveCount
   2000         0.923 ms         25373         43953
  6000         1.89 ms         80749         145506
  10000         2.974 ms         152574         261453
  14000         4.168 ms         234913         380111
  18000         5.193 ms         290001         472045
  22000         6.498 ms         388232         627814
  26000         7.485 ms         448619         712667
  30000         8.743 ms         520954         838309
------------------------------------------------------------
Part D: Time Analysis of Radix Sort
Array Size      Time Elapsed
   2000         0.917 ms
  6000         1.885 ms
  10000         2.972 ms
  14000         4.176 ms
  18000         5.215 ms
  22000         6.487 ms
  26000         7.494 ms
  30000         8.744 ms
```

# Question 3

## Graph for Sorting Algorithms



## Comments on Results

Quick, merge and radix sort is usually faster than the insertion sort. This is what I expected. Insertion sort was the slowest, radix sort was the fastest. Running times of quick and radix sort are close to each other.

Due to the time complexity of the insertion sort, there is an exponential increase which I can see on the graph.

There was no difference with my experiment and the theoretical values. But I expected radix sort to be faster but it is very close to other sorting algorithms. This can be because there was not much digits in numbers.

If I applied this sorting algorithms to an array with increasing numbers, there should be significant decrease in the times of quick sort. Because the worst case of the quicksort is in already sorted arrays. Other sorting algorithms would be similar, I guess.