CS325

Guangyu Zhang

1.

a).

Solve this problem in divide and conquer way can follow the steps: first, we divide the whole array into 2 parts as subproblems, and recursively divide the array. Then, conquer, for each subproblem, find its max or min value. Finally, combination, compare the results of each subproblem to find the max or min.

Pseudo-code:

```
Find max_or_min(array[n])
{
if (len(array[n])) = 1
return array[0]
else
divide_to_2(array[n])
sub_a=array[0, n/2]
sub_b=array[n/2, n]
Find max_or_min(sub_a)
Find max_or_min(sub_b)
Compare (Find max_or_min(sub_a), Find max_or_min(sub_b))
return max, min
}
```

b).

$$T(n) = 2T(n/2) + c$$

c).

$$T(n) = 2T(n/2) + c$$

$$a = 2, b = 2, f(n) = c$$

$$case1: f(n) = c = n^0 = \log_2 2 - 1\,, \varepsilon = 1 > 0,$$

$$T(n) = \theta(n)$$

The iterative algorithm should be:

For each number, compare to n – 1 others numbers, so it has 2 for loops.

$$T(n) = \theta(n^2)$$

Therefore, the recursive min_and_max algorithm has less time consumption than iterative algorithm.

2.

a).

```
Mergesort3(array[n])
{
m = n/3
array_a = array[0, m]
array_b = array[m, 2m]
array_c = array[2m, n]
Mergesort3(array_a)
Mergesort3(array_b)
Mergesort3(array_c)
}

Merge3()
for (i=0; i < m; i++){
result[] = array[i]
}
for (i=0; i < 2m-m; i++){
```

```
result[] = array[i]
}
for (i=0; i < n; i++){
result[] = array[i]
}
```

b).

$$T(n) = 3T(n/3) + n$$

c).

$$T(n) = 3T(n/3) + n$$

$$a = 3, b = 3, f(n) = n$$

$$case1: f(n) = c = n^1 = \log_3 3,$$

$$T(n) = \theta(nlgn)$$

3. submitted as code

4.

a). submitted as code

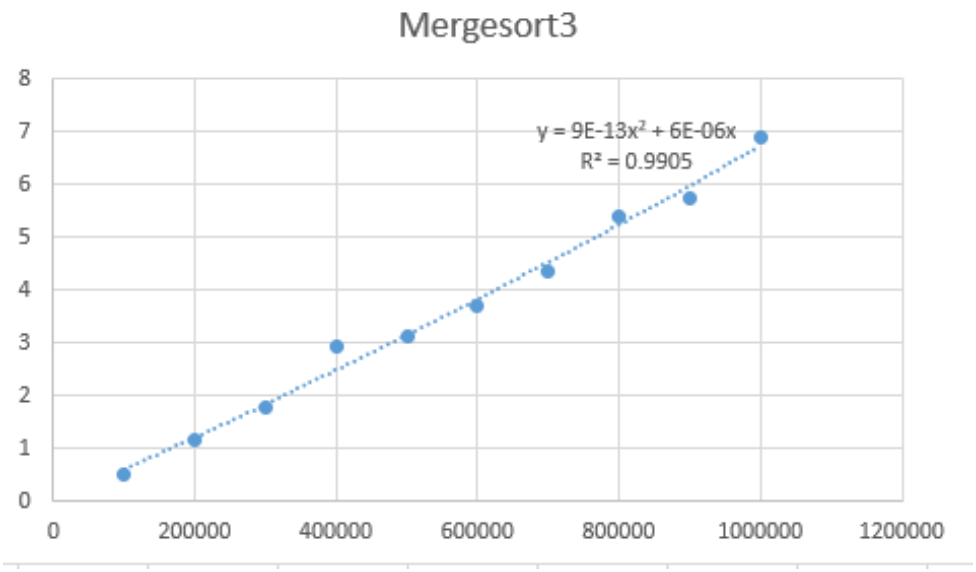
b).

Number(n)	merge sort 3
100000	0.51265549659729 sec
200000	1.1659111976623535 sec
300000	1.7732818126678467 sec
400000	2.9261486530303955 sec

500000	3.12766432762146 sec
600000	3.706111192703247 sec
700000	4.340397119522095 sec
800000	5.410558223724365 sec
900000	5.754785060882568 sec
1000000	6.912974834442139 sec

c).

the best type is polynomial, since $T(n) = \theta(nlgn)$



d).

3-way merge sort runs a little faster than 2-way merge sort, but generally they have similar time consumption. The theoretical running times of 3-way and 2-way merge sort are same: $T(n) = \theta(nlgn)$. However, in my experiment, 3-way merge sort is a little faster than 2-way merge sort. One possibility is that my log part causes this difference.