Algorithms Elyas Ata

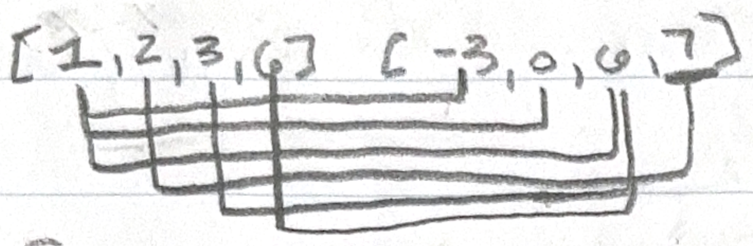
1.) Mergesort [1,2,3,6] and [-3,0,6,7]

our list is already sorted so we will merge.

we will make a new list to store our comparisons

[1,2,3,6] [-3,0,6,7]

we will compare our left list to right



[-3,0,1,2,3,6,6,7]

we initially compare then insert

1 and -3. since 1 > -3 we insert

-3 into new array then move down.

1 > 0 so 0 is next. 1 < 6, 2 < 6,

3 < 6, 6 = 6, 6 < 7

2. Insertion Sort algorithm

[-21,5,7,-10,61,8,3,10]

when comparing we start left-most side and compare

we know -21 < 5 so keep.

[-21,5,7,-10,61,8,3,10]

[-21,5,7] 5 < 7 keep

(-21 < 5, 5 < 7)

[-21,5,7,-10]

(-21 < -10, but -10 < 5 so insert.)

[-21,-10,5,7]

[-21,-10,5,7,61] < 61 greater than so keep at end.

[-21,-10,5,7,61,8] 8 < 61 insert 8

[-21,-10,5,7,8,61]

[-21,-10,5,7,8,61,3,10] 3 is < than 5,7,8,61, insert

[-21,-10,3,5,7,8,61,10] 10 is < 61 so insert 10

[-21,-10,3,5,7,8,10,61] before 61.

Problem 3: quicksort

[-5, 4, 2, 619, 11, 5, 620, -3]

my pivot will be 5

$S_1 \leq p$ = [-5, 4, 2, -3]

$S_2 > p$ = [619, 11, 620]

for S_1 partition will be 2

S_1 = [-5, 4, 2, -3]

p=2

S_1 = [-5, -3] S_2 = [4]

[-5, -3, 2, 4, 5]

S_2 = [619, 11, 620]

p = 619

S_1 = 11 S_2 = 620

[11, 619, 620]

Concat

[-5, -3, 2, 4, 5, 11, 619, 620]

problem 4: Shellsort

[5, 10, 60, 0, -1, 34, 6, 10]

8/2 = gap 4

[5, 10, 60, 0, -1, 34, 6, 10]
1 2 3 4 1 2 3 4

iteration 1: yes (5, -1) -1, 5 insert -1, 10, 60, 0, 5, 34, 6, 10

2: no 10, 34

3: yes (60, 6) 6, 60 -1, 10, 6, 0, 5, 34, 60, 10

4: no (0, 10)

gap 2

[-1, 10, 6, 0, 5, 34, 60, 10]

sublist 1: -1, 6, 5, 60 2: 10, 0, 34, 10 0, 10, 10, 34

-1, 10, 5, 0, 6, 34, 60, 10

iteration 2: -1, 0, 5, 10, 6, 10, 60, 34

A = -1, 0, 5, 6, 10, 10, 34, 60

Problem 5: Rankings

fastest → slowest

1. Merge sort - O(nlogn): any list that will always split into 2 list. has a consistent speed on any dataset.

2.) QuickSort - O(nlogn): S_1 or S_2 during best case will be equal size only one pass through array.

3.) Shell Sort - $O(n\log^2 n)$: reduces comparisons

4.) insertion sort - O(n): sorted input, best case it'll go through 1 comparison in each pass.

5.) Selection Sort - best and worst case $O(n^2)$ it will always do 2 passes through the array.

6.) Bubble sort - O(n): Sorted input, it'll have n-1 comparisons, no swaps. Worst case $O(n^2)$ max swaps.

9.) Bubble and Selection sort take all the room which makes the faster algorithms look invisible or squished in a sense. I changed the y-axis to log scale which can be visualized

10.) Bubble and selection perform for small arrays but when it came towards larger it became super slow. runtimes grow dramatically.

Insertion was on the upper side but due to fewer swaps runtime didn't grow as dramatic.

Shell, Merge, and quick all met expectations as nlogn increased scaled efficiently

yes, my prediction matched the algorithm analysis.