Sorting and Searching Techniques

18 March 2024 14:07

Step-by-step procedures defining a set of instructions to solve problems

Good Algorithm is: (i) Produce correct outputs for any set of legal inputs
(ii) Execute efficiently with fewest number of steps possible
(iii) Easy to understand & follow
(iv) Easy to modify

BUBBLE SORT

Time Complexity - O(n2)

(I)
7 6 4 3
6 7 4 3
6 4 7 3
6 4 3 7

(II)
6 4 3 7
4 6 3 7
4 3 6 7

(III)
4 3 6 7
3 4 6 7
yay!

(I)
5 1 6 2 4 3
1 5 6 2 4 3
1 5 2 6 4 3
1 5 2 4 6 3
1 5 2 4 3 6

(II)
1 5 2 4 3 6
1 2 5 4 3 6
1 2 5 3 4 6

(III)
1 2 5 3 4 6
1 2 3 5 4 6

(IV)
1 2 3 5 4 6
1 2 3 4 5 6

(I)
54 26 93 17 77 31 44 55 20
26 54 93 17 77 31 44 55 20
26 54 17 93 77 31 44 55 20
26 54 17 77 93 31 44 55 20
26 54 17 77 31 93 44 55 20
26 54 17 77 31 44 93 55 20
26 54 17 77 31 44 55 93 20
26 54 17 77 31 44 55 20 93

(II)
26 54 17 77 31 44 55 20 93
26 17 54 77 31 44 55 20 93
26 17 54 31 77 44 55 20 93
26 17 54 31 44 77 55 20 93
26 17 54 31 44 55 77 20 93
26 17 54 31 44 55 20 77 93

(III)
26 17 54 31 44 55 20 77 93
17 26 54 31 44 55 20 77 93
17 26 31 54 44 55 20 77 93
17 26 31 44 54 55 20 77 93
17 26 31 44 54 20 55 77 93

(IV)
17 26 31 44 54 20 55 77 93
17 26 31 44 20 54 55 77 93

(V)
17 26 31 44 20 54 55 77 93
17 26 31 20 44 54 55 77 93

(VI)
17 26 31 20 44 54 55 77 93
17 26 20 31 44 54 55 77 93

(VII)
17 26 20 31 44 54 55 77 93
17 20 26 31 44 54 55 77 93

SELECTION SORT

Time Complexity - O(n2)

7 1 4 3 2 → find the smallest & swap!

1 7 4 3 2

1 2 4 3 7

1 2 3 4 7 Sorted!

45 12 78 32 19 65 27 91 53 10

10 12 78 32 19 65 27 91 53 45

10 12 19 32 78 65 27 91 53 45

10 12 19 27 78 65 32 91 53 45

10 12 19 27 32 65 78 91 53 45

10 12 19 27 32 45 78 91 53 65

10 12 19 27 32 45 53 91 78 65

10 12 19 27 32 45 53 65 78 91

INSERTION SORT

Time Complexity - O(n2)

4 3 2 10 12 1 5 6

3 4 2 10 12 1 5 6

2 3 4 10 12 1 5 6

1 2 3 4 10 12 5 6

1 2 3 4 5 10 12 6

1 2 3 4 5 6 10 12

32 15 7 44 19 51 23 8 40 12

15 32 7 44 19 51 23 8 40 12

7 15 32 44 19 51 23 8 40 12

7 15 19 32 44 51 23 8 40 12

7 15 19 23 32 44 51 8 40 12

7 8 15 19 23 32 44 51 40 12

7 8 15 19 23 32 40 44 51 12

7 8 12 15 19 23 32 40 44 51 YAY!

54 29 11 67 36 82 19 45 63 27

29 54 11 67 36 82 19 45 63 27

11 29 54 67 36 82 19 45 63 27

11 29 36 54 67 82 19 45 63 27

11 19 29 36 54 67 82 45 63 27

11 19 29 36 45 54 67 82 63 27

11 19 29 36 45 54 63 67 82 27

11 19 29 36 45 54 67 82 63 27

11 19 29 36 45 54 62 67 82 27

11 19 27 29 36 45 54 63 67 82

---

* Differention between Bubble Sort & Insertion Sort -

INSERTION SORT	BUBBLE SORT
① Left Direction ←	① Right Direction →
② 1 & 0 are compared → 2 & 1 are compared & so on	② 0 & 1 are compared → 1 & 2 are compared & so on
③ Done in one pass of list	③ Done in multiple passes of list
④ Less time & less caps	④ More time
⑤ Time Complexity - O(n2)	⑤ Time Complexity - O(n2)

DIVIDE & CONQUER

Merge Sort Quick Sort Binary Search

Searching Techniques

Sequential Search Interval Search

Fundamentals of D & C Strategy -

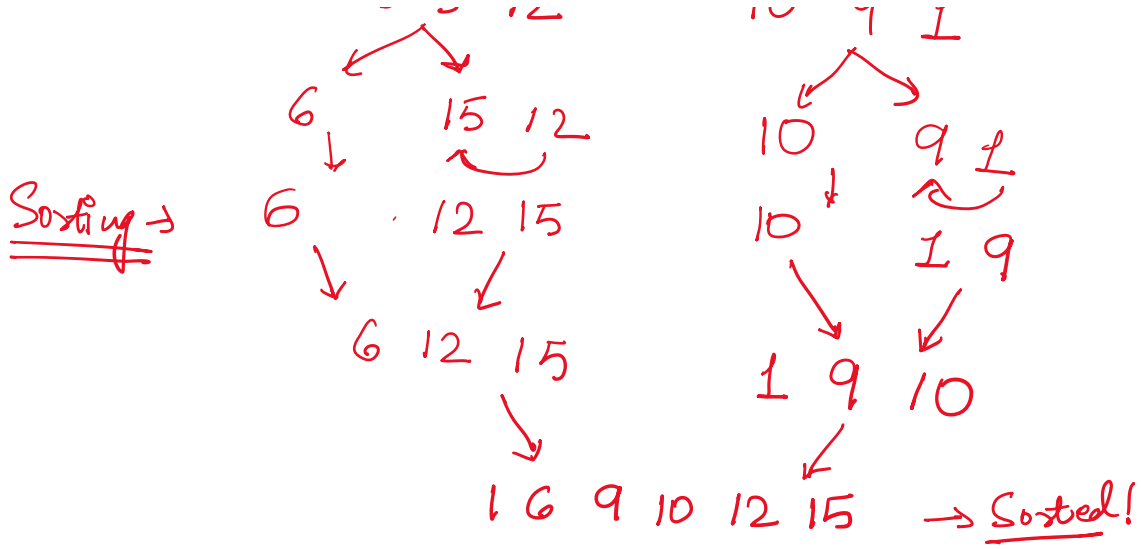
→ Relational Formula

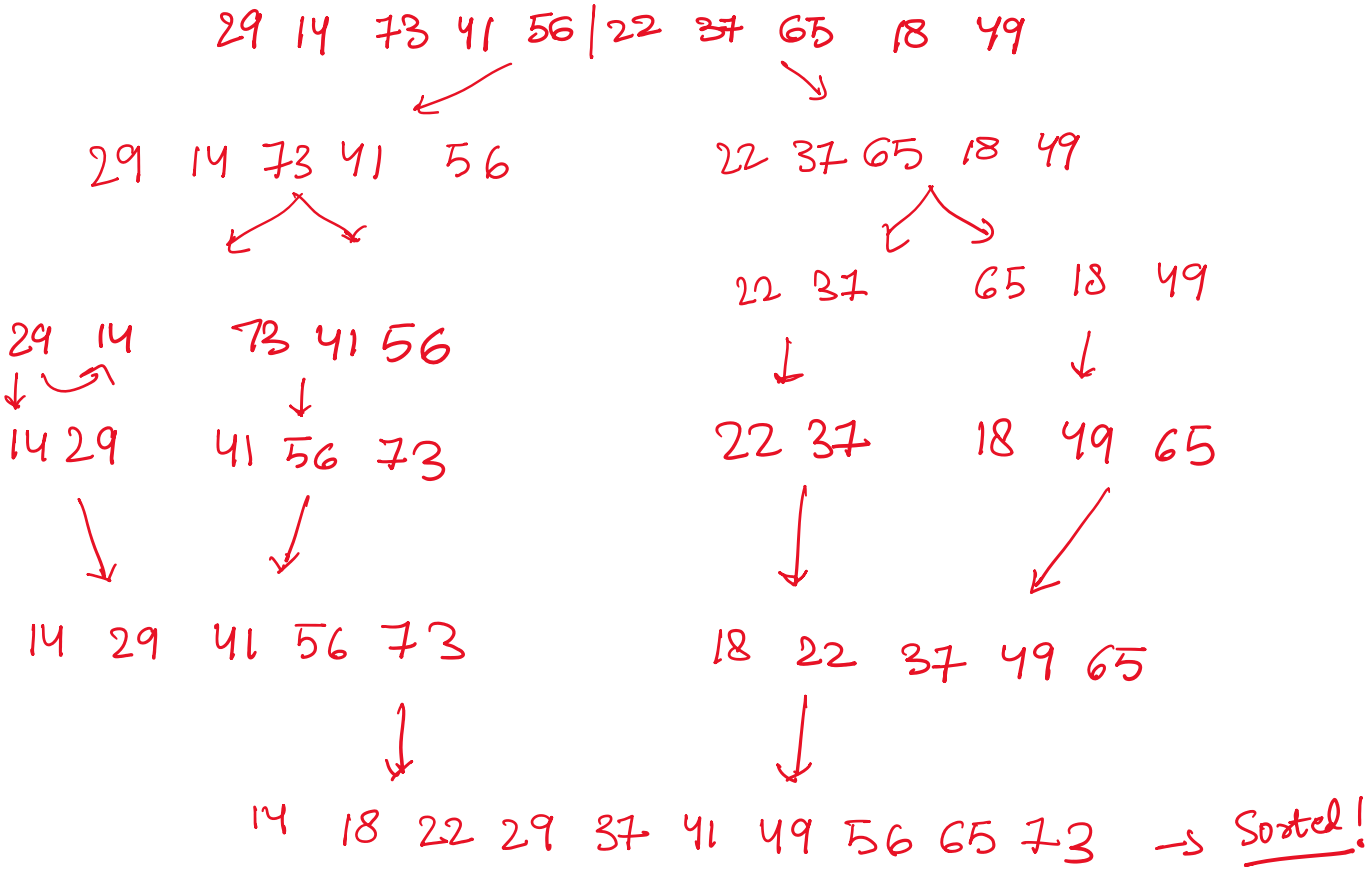
→ Stopping Condition

MERGE SORT

6 5 12 | 10 9 1

6 5 12 10 9 1

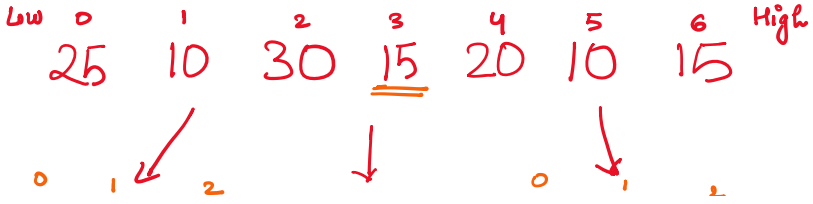




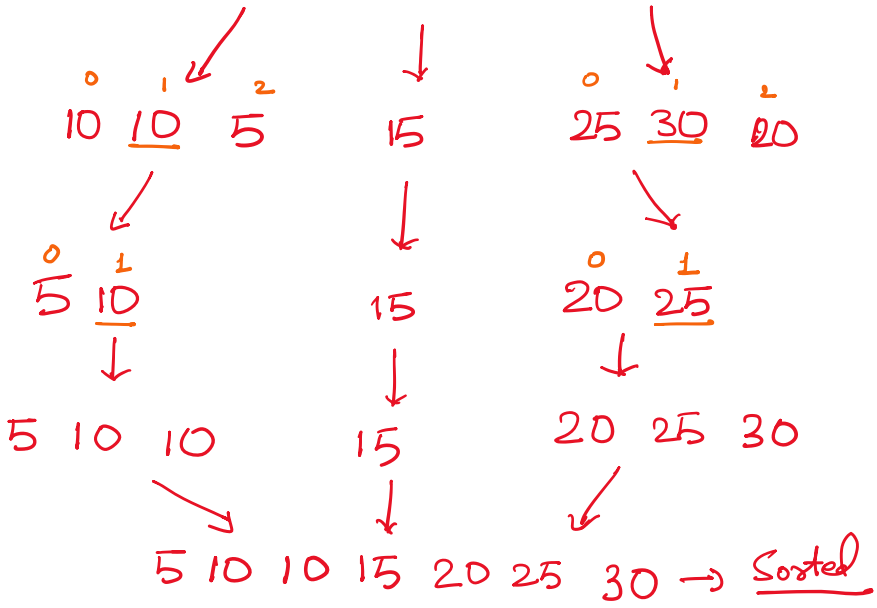
QUICK SORT

Time Complexity – O(n log n)

1st Pivot Index – $0 + \frac{(6-0)}{2} = 3$



2nd Pivot Index –



2nd Pivot Index - $\frac{3}{2} = 1.5$

3rd Pivot Index - $\frac{2}{2} = 1$

Types of Analysis -

(i) Worst Case	Maximum no. of steps
(ii) Best Case	Minimum no. of steps
(iii) Average Case	Average no. of steps

BIG O NOTATION – Provides the upper bound or worst-case scenario for the algorithm.

TIME COMPLEXITY

(i) O(1)	Execute in same time or space regardless of size of input data.
(ii) O(log n)	Performance based directly on the logarithm of size of input data.
(iii) O(n)	Performance will grow linearly or directly proportional to size of input data.
(iv) O(n log n)	Performance is based on the input of its logarithm.
(v) O(n2)	Performance will depend directly on square size of input data.
(vi) O(2n)	Run-time doubles with each input element in the algorithm.

SPACE COMPLEXITY

- Refers to the amount of resources taken by an algorithm to complete the operation on an instance with respect to the size of its input.

BINARY SEARCH (only works on sorted list)

Time Complexity - O(log n)

Find 23

2 5 8 12 16 | 23 38 56 72 91

2 5 8 12 16 23 38 56 72 91

2 5 8 12 16 23 | 38 56 72 91

2 5 8 12 16 (23) 38 56 72 91

SORTING TECHNIQUES COMPARED

CRITERIA	BUBBLE SORT	SELECTION SORT	INSERTION SORT	MERGE SORT	QUICK SORT
Time complexity (Best to worst)	O(n)	O(n2)	O(n)	O(n log n)	O(n log n)
	O(n2)	O(n2)	O(n2)	O(n log n)	O(n log n)
	O(n2)	O(n2)	O(n2)	O(n log n)	O(n2)
space Complexity	O(1)	O(1)	O(1)	O(1)	O(1)

① Sorting Arrays

② Differentiate btw Sorts

③ Advantages & Disadvantages
④ Algorithms

BUBBLE SORT

(i) 10, 12, 15, 70, 28, 35, 7

I) 10 12 15 70 28 35 7
10 12 15 28 70 35 7
10 12 15 28 35 70 7
10 12 15 28 35 7 70

II) 10 12 15 28 35 7 70
10 12 15 28 7 35 70

III) 10 12 15 7 28 35 70

IV) 10 12 7 15 28 35 70

V) 10 7 12 15 28 35 70

VI) 7 10 12 15 28 35 70

(ii) 10, 20, 35, 2, 72, 80, 29

I) 10 20 35 2 72 80 29
10 20 2 35 72 80 29
10 20 2 35 72 29 80

II) 10 20 2 35 72 29 80
10 2 20 35 72 29 80
10 2 20 35 29 72 80

III) 2 10 20 35 29 72 80
2 10 20 29 35 72 80

(iii) 80, 81, 42, 38, 26, 13, 67

I) 80 42 81 38 26 13 67
80 42 38 81 26 13 67
80 42 38 26 81 13 67
80 42 38 26 13 81 67
80 42 38 26 13 67 81

II) 80 42 38 26 13 67 81
42 80 38 26 13 67 81
42 38 80 26 13 67 81
42 38 26 80 13 67 81
42 38 26 13 80 67 81
42 38 26 13 67 80 81

III) 42 38 26 13 67 80 81
38 42 26 13 67 80 81
38 26 42 13 67 80 81

IV) 38 26 13 42 67 80 81
26 38 13 42 67 80 81

38 26 42 13 67 80 81

38 26 13 42 67 80 81

26 38 13 42 67 80 81

26 13 38 42 67 80 81

(iv) 13 26 28 42 67 80 81

(iv) Time Complexity of Bubble Sort -

Best Case - $O(n)$

Worst Case - $O(n^2)$

Average Case - $O(n^2)$

(iv) Space Complexity of Bubble Sort - $O(1)$

(v) Advantages & Disadvantages of Bubble Sort

→ Advantages - Simple & easy to implement

- Takes less space as the items are swapped in their place

Disadvantages - Not suitable for large datasets

- Inefficient as compared to other sorting methods

(vi) Algorithm for Bubble Sort -

Step1: Start

Step2: Compare the first element with the second element & if first element is greater than second element, swap.

Step3: Continue the same step with the second & third element.

Step4: Repeat steps 3 & 4 until the largest element is traversed to the end of the array/list. First Pass completed

Step5: Repeat Steps - 3, 4 & 5 until the array is sorted.

Step6: End/Stop

SELECTION SORT

(i) 7, 9, 27, 4, 32, 44

(I) 7 9 27 4 32 44

4 9 27 7 32 44

* swap the smallest no. with the first element

(II) 4 9 27 7 32 44

4 7 27 9 32 44

* swap the smallest digit with second element
* keep repeating the steps until the array is sorted!

(III) 4 7 27 9 32 44

4 7 9 27 32 44

(ii) 47, 2, 46, 59, 78, 19

(I) 47 2 46 59 78 19

2 47 46 59 78 19

* swap the smallest digit with the first element

(II) 2 47 46 59 78 19

2 19 46 59 78 47

* take the second element & swap it with the next smallest digit from the right

(III) 2 19 46 59 78 47

2 19 46 47 78 59

* keep repeating until the array is sorted

(IV) 2 19 46 47 78 59

2 19 46 47 59 78

(iii) Time Complexity of Selection Sort -

Best Case - $O(n^2)$

Worst Case - $O(n^2)$

Average Case - $O(n^2)$

(iv) Space Complexity of Selection Sort - $O(1)$

(v) Advantages & Disadvantages of Selection Sort -

Advantages - Simple & easy to implement
- Suitable for small datasets
- more efficient than bubble sort

Disadvantages - not suitable for large datasets

Disadvantages - not suitable for large datasets
- poor time complexity

(vi) Algorithm for Selection Sort

Step 1 : Start

Step 2 : Select the first element of the array & swap it with the smallest element to its right

Step 3 : Now select the second element & repeat Step 2.

Step 4 : Continue Step 2 with all the next elements in the list until its completely sorted.

Step 5 : Stop / End

INSERTION SORT

(i) 4, 50, 49, 17, 23, 48

(I) 4 50 49 17 23 48 * no swap coz 4 < 50

(II) 4 50 49 17 23 48
4 17 49 50 23 48 * 17 inserted btw 4 & 50

(III) 4 17 49 50 23 48
4 17 23 49 50 18 * 23 inserted btw 17 and 49

(IV) 4 17 23 49 50 18
4 17 18 23 49 50 * 18 inserted btw 17 and 23; array sorted!

(ii) 42, 58, 39, 76, 18, 29

(I) 42 58 39 76 18 29 * no swaps

(II) 42 58 39 76 18 29
39 42 58 76 18 29 * 39 inserted before 42 bcoz 39 < 42 and 39 < 58

(III) 39 42 58 76 18 29

(iii) 39 42 58 76 18 29 — 39 < 58

18 39 42 58 76 29 * 18 inserted before 39

(iv) 18 39 42 58 76 29

18 29 39 42 58 76 * 29 inserted btw 18 and 39; Sorted!

(iii) Time Complexity of Insertion Sort

Best Case - O(n)

Worst Case - O(n2)

Average Case - O(n2)

(iv) Space Complexity of Insertion Sort - O(1)

(v) Advantages of Disadvanges of Insertion Sort

Advantages - Simple & easy to implement
- efficient for small datasets
- doesn't take up much space

Disadvantages - requires more time
- not suitable for large datasets

(vi) Algorithm for Insertion Sort

Step1 : Start

Step2 : Compare element 1 with element 0, if 1 > 0, swap them.

Step3 : Now traverse to the next element, i.e. 2, and compare 2 with all the elements to its left (i.e 0 and 1) and if 2 < 1 and 2 < 0, insert it before the element it is smallest from; Else don't swap

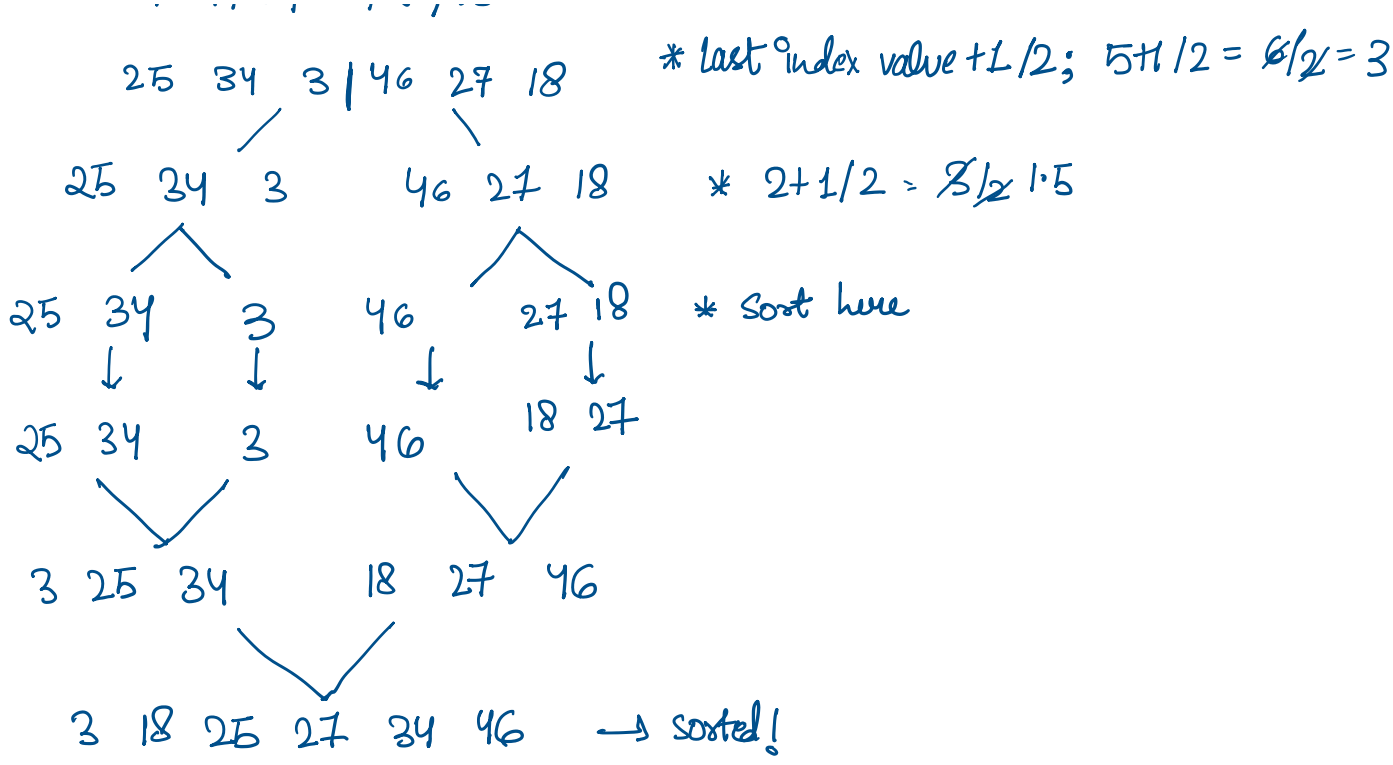
Step4 : Repeat the above steps with all the elements until the array is completely sorted.

Step5 : Stop/End

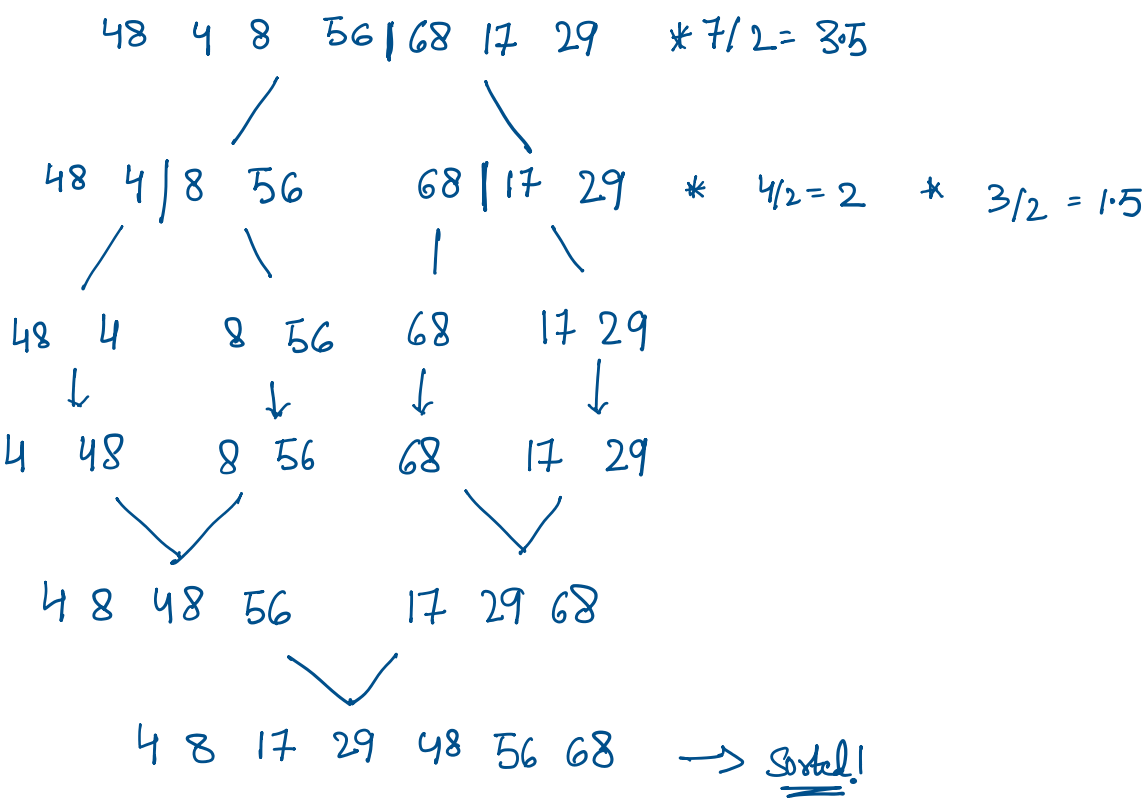
MERGE SORT

i) 25, 34, 3, 46, 27, 18

25 34 3 | 46 27 18 * last index value +1 /2; 5+1 /2 = 6/2 = 3



(ii) 48, 4, 8, 56, 68, 17, 29



(iii) Time Complexity of merge Sort -

Best Case - O(n log n)

Average Case - O(n log n)

Worst Case - O(n log n)

(iv) Space Complexity of Merge Sort - O(n)

(iv) Space Complexity of Merge Sort - O(n)

(v) Advantages & Disadvantages of Merge Sort -

Advantages - Better for handling sequential access files
- Best case for handling slow access data like tape drives

Disadvantages - Takes additional memory space for the temporary array
- Will go through the whole process even if the array is sorted
- Slower as compared to other sorting algorithms for smaller tasks

(vi) Algorithm for Merge Sort

Step 1 : Start

Step 2 : Divide the array into two parts using - $\left[\frac{\text{last index value} + 1}{2}\right]$

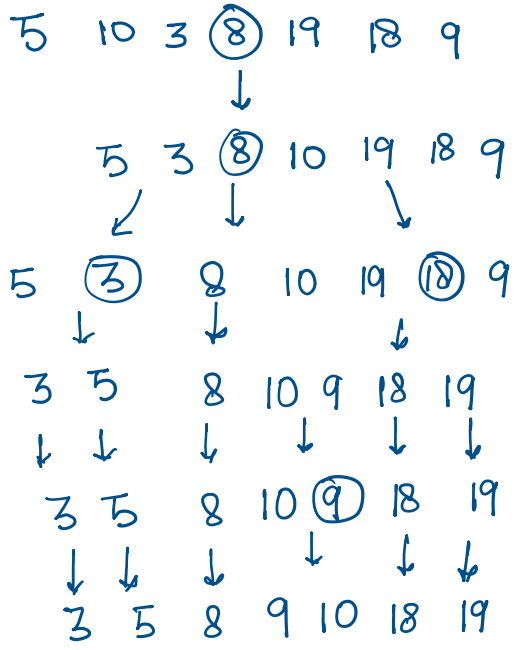
Step 3 : Keep dividing the arrays until the elements are in a set of 2 (or one if there's an odd number of elements)

Step 4 : Now sort and merge these data elements, until you get the whole sorted array. (from smallest to greatest)

Step 5 : Stop

QUICK SORT

i) 5, 10, 3, 8, 19, 18, 9



* Pivot = 8

* Divide the list with elements greater than pivot on right & lesser than pivot on left

* keep repeating the steps

→ sorted!

3 5 8 9 10 18 19 → sorted!

(ii) 19, 17, 15, 12, 16, 18, 4, 11, 13

19 17 15 12 16 18 4 11 (13) * Pivot = 13

12 4 11 (13) 19 17 15 16 18

12 4 (11) 13 19 17 15 16 (18)

4 11 12 13 15 16 18 19

4 11 12 13 15 16 18 19 → sorted!

(iii) Time Complexity of Quick Sort

Best Case - O(n log n)
Average Case - O(n log n)
Worst Case - O(n2)

(iv) Space Complexity of Quick Sort - O(n)

(v) Advantages & Disadvantages of Quick Sort -

Advantages - Extremely fast
- Gives good result when array is in random order
- Works well with small, medium & large datasets

Disadvantages - Very complex
- Very poor time complexity in worst-case scenario

(vi) Algorithm for Quick Sort

Step 1 : Start

Step 2 : Select pivot (either given, or last element or middle element)

Step 3 : Keep pivot in the middle with the elements greater to it on the right side and the elements lesser than it on the left side

Step 4 : Now apply the same for the left and right side of the pivot as well

right side and the elements lesser than it on the left side

Step 4 : Now divide the array from the left & right side of pivot and Repeat Steps 2 and 3 until the whole array is sorted.

Step 5 : Stop/End

LINEAR SEARCH

(i) 2, 5, 10, 12, 18, 60

Search - 10

$2 \neq 10$, $5 \neq 10$, $10 = 10$!

(ii) Time Complexity of Linear Search -

Best Case - O(1)

Worst Case - O(n)

(iii) Space complexity of Linear Search - O(1)

(iv) Advantages & Disadvantages of Linear Search

Advantages - Small to medium arrays can be searched easily
- array doesn't need to be sorted
- insertion & deletion in an array won't affect linear search

Disadvantages - not suitable for large datasets
- speed limitation

(v) Algorithm for Linear Search

Step 1 : Start

Step 2 : Compare the first element with element to be searched for

Step 3 : Keep comparing all the elements in the list until the element to be searched has been found

Step 4 : Stop/End

BINARY SEARCH

(i) 5, 10, 15, 20, 25, 30

Search - 25

5 10 15 20 25 30 - 5+1/2 = 3 ; 20 ≠ 25

20 (25) 30 - 2+1/2 = 1.5 ; 25 = 25 Match found!

(ii) 2, 8, 10, 11, 15, 16, 17, 19

Search - 8

2 8 10 11 15 16 17 19 - 7+1/2 = 4 ; 15 ≠ 8

2 8 10 11 15 - 4+1/2 = 2.5 ; 10 ≠ 8

2 (8) 10 - 2+1/2 = 1.5 ; 8 = 8 Match found!

(iii) Time Complexity of Binary Search

Best Case - O(1)

Worst Case - O(log n)

(iv) Space Complexity of Binary Search - O(1)

(v) Advantages & Disadvantages of Binary Search

Advantages - Eliminates half the list through divide & conquer
- faster than linear search
- suitable for large datasets

Disadvantages - Difficult as compared to linear search
- cannot work on unsorted sets

(vi) Algorithm for Binary Search

Step 1: Start

Step 2: Find out the middle element of the list by (last index value + 1 / 2) ; if

Step 3: If mid ≠ element to be searched, eliminate the side of the array that is greater than the element to be searched.

Step 4: Keep repeating Steps 2 and 3 until mid = element to be searched

Step 5: Stop/End