

defined in
#include <algorithm>

Sorting

- `sort()` provide sorting of the data stored in containers which allow random access.
(Array, vector, deque) → accessing in constant time.

- For Array → points to element after last index

→ `sort(a, a+n)`

↑
points to first index

→ `sort(a, a+n, greater<int>())`

↑
comparison function

- For vector

→ `sort(v.begin(), v.end());`

→ it bracket मतलब नहीं

→ `sort(v.begin(), v.end(), greater<int>());`

→ You can create your own Comparison function:

```
struct Point
```

```
{ int x, y;
```

```
};
```

```
bool mycomp(Point p1, Point p2)
```

```
{ return (p1.y > p2.y)
```

```
}
```

```
int main()
```

```
{
```

```
    Point a[] = {{3, 10}, {2, 8}, {5, 4}};
```

```
    sort(a, a+3, mycomp);
```

```
    for(auto i: a)
```

```
        cout << i.x << " " << i.y << endl;
```

```
}
```

O/p → 3 10

2 8

5 4

जिसका y बड़ा वो सर

(decreasing order of y value)

- Java sort function uses QuickSort, InsertionSort, Merge Sort
- Python ————— Tim Sort (Merge Sort + Insertion Sort)

Worst & Average case $O(n \log n)$
 C++ uses IntroSort (Hybrid of QuickSort, HeapSort, InsertionSort)

- By default it uses Quick Sort, if QS is doing n^2 operations and taking more than $(n \log n)$ time it switches to HeapSort and if Array size is very small it switches to Insertion Sort.

→ Stability in Sorting Algorithm :

- if two elements have same value, then they should appear in the same order as they appear in the original Array.

- Stability is important when we have elements like this
 eg. { ("Anil", 50), ("Ayan", 80), ("Piyush", 50), ("Ramesh", 80) }

~~{ ("Anil", 50), ("Ayan", 80),~~

{ ("Anil", 50), ("Piyush", 50), ("Ayan", 80), ("Ramesh", 80) }

→ Defn of a stable sort algorithm

- Stability is checked when we have more than one element in a object.

O/p → { ("Piyush", 50), ("Anil", 50), ("Ayan", 80), ("Ramesh", 80) }

→ Stable Algo

- unstable Algo doesn't care of the order

→ Eg of Stable Sorts: Bubble, Insertion, Merge

→ Eg of Unstable Sort: Selection, QuickSort, HeapSort

SORTING METHODS

(i) Bubble Sort:

In this sorting algorithm elements are compared with the next adjacent elements and if required we need to swap the elements as per the condition.

eg:

0	1	2	3	4	5
4	3	1	8	6	2

Sorting in ascending order

if $(4 > 3)$ then swap

3	4	1	8	6	2
---	---	---	---	---	---

$4 > 1$ swap

3	1	4	8	6	2
---	---	---	---	---	---

$4 \not> 8$ no swapping

$8 > 6$ swap

3	1	4	6	8	2
---	---	---	---	---	---

$8 > 2$ swap

3	1	4	6	2	8
---	---	---	---	---	---

→ further repetition 321 swap ---

एक बार जब अंदर वाला for loop complete होगा तो यही तक चलेगा।

*** एक बार जब अंदर वाला for loop complete हो जाएगा तो जो सबसे largest number होगा वो आखिरी में चले जाएगा (here 8)

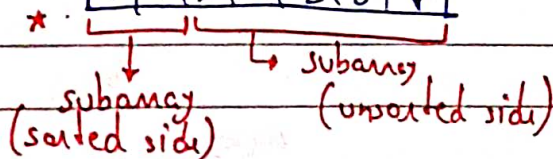
(1) Selection Sort

- The selection sort algorithm sorts an array by repeatedly finding the minimum element from unsorted part and putting it at the beginning. or vice-versa
- The algorithm maintains two subarrays in a given array.
 - 1) The subarray which is already sorted.
 - 2) Remaining subarray which is unsorted.
- In every iteration of selection sort, the minimum element from the unsorted subarray is picked and moved to the sorted subarray.

[8 | 5 | 1 | 6 | 2 | 0 | 7]

after one iteration [0 | 5 | 1 | 6 | 2 | 8 | 7]

after 2nd " [0 | 1 | 5 | 6 | 2 | 8 | 7]

* 

subarray (sorted side) subarray (unsorted side)

- एक बार जब for loop चल जाएगा तब वहां, तब minimum element 1st position पर आ जाएगा and so on

- Selection Sort less memory width compared to Quick Sort, Merge Sort, Insertion Sort etc. But Bubble Sort is optimal in terms of memory width.
- Not Stable, In-place (doesn't use extra space)
- $O(n^2) \rightarrow$ always
- Basic Idea for heap sort

→ Programme
main()
{

int i, j, A[1000], n, t, min, k;

same here → ①

for(i=0; i<n-1; i++)
{

min = A[i];

for(j=i+1; j<n; j++)
{

if(min > A[j])
{

k = j;

min = A[j];

A[k] = min;

}

if(A[i] != min) → swapping condition
{

t = A[i];

A[i] = A[k];

A[k] = t;

} swap condition
is done

}

printf("Sorted array is \n");

for(i=0; i<n; i++)

printf("%d ", A[i]);

}

Output: enter the size of array
enter 8 integers 6 5 7 4 8 3 9 1
Sorted array is
1 3 4 5 6 7 8 9

- in some case the elements are at the right place so don't need to swap.
- we will swap only if ^{min} element is not at right place.

(iii) Insertion Sort

- it is based on the principle that a new key K is inserted at its appropriate position in an already sorted sub list.

- say List $L = \{K_1, K_2, K_3, \dots, K_{n-1}, K_n\}$

→ in first pass K_2 compared to K_1 , if not at appropriate place swap

→ in second pass K_3 compared to K_1 , if ^{condition} true swap then

say $\{K_3, K_2, K_1, \dots\}$

and also K_1 compared to K_2 , if condition true swap

say $\{K_3, K_1, K_2, \dots\}$

and so on

eg: $\{16, 36, 4, 22, 100, 1, 54\}$

after 1st pass → $\{[16, 36], 4, 22, 100, 1, 54\}$
 ^{it is compared}

after 2nd pass → $\{[4, 16, 36], 22, 100, 1, 54\}$
 ^{sorted side}

after 3rd pass → $\{[4, 16, 22, 36], 100, 1, 54\}$

after 4th " → $\{[4, 16, 22, 36, 100], 1, 54\}$

after 5th " → $\{[1, 4, 16, 22, 36, 100], 54\}$
 ^{sorted list}

after 6th " → $\{1, 4, 16, 22, 36, 54, 100\}$

Bubble Sort \rightarrow Adjacent element compare
 Selection Sort \rightarrow Subarray sorted & inserted
 Insertion Sort \rightarrow Key at correct position

classmate

Date
Page

(I) Insertion Sort:

- it is in-place (no extra space)
- Stable
- ✓ It is most effective if array size is small
- Time Sort (Merge Sort + Insertion Sort)

```
void insertionSort(int a[], int n)
{
```

```
    for (int i = 0; i < n; i++)
    {
```

```
        int key = a[i];
```

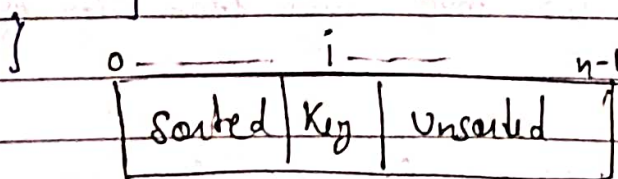
```
        int j = i - 1;
```

```
        while (j >= 0 && a[j] > key)
        {
```

```
            a[j+1] = a[j];
```

```
            j--;
```

```
        }
        a[j+1] = key;
```



Worst $\rightarrow O(n^2)$, Best $\rightarrow O(n)$

eg: 5, 20, 40, 60, 10, 30 say $i = 4$
 0, 1, 2, 3, 4, 5
 ← Sorted →

5, 20, 40, 60, 10, 30
 ← j i/key

Merge Sort:

- it is an Divide & Conquer Algorithm
(divide the array in two parts, sort these two parts, then merge these two parts)
- Stable
- $O(n \log n) \leftarrow TC$ | $O(n) \leftarrow SC$ (not in-place)
- Block Merge Sort:
- It is a sorting algo combining atleast two merge operations with an insertion sort to arrive at $O(n \log n)$ in-place stable sorting algo.
- For Array Quick Sort better than MS
- For linked list MS better than $O(1)$ Aux space
- Well suited for external sorting (bring parts of array in RAM then sort)