- Selection Sort

```
void  Selection_Sort ( int* start,  int* end )
{
    for ( int x = 0, min = 0;  x < (end-start)-1;  ++x )
    {
        for ( int y = (min=x , (x+1));  y < end-start ;  ++y )
        {
            if ( start[y] < start[min] )
            {
                    min = y ;
            }
        }
        swap ( start[x], start[min] );
    }
}
```

Most Optimized!
By,
THE MR

- In my selection sort algorithm, for the Array :

→ A[] = { 15, 2, 8, 4, 1, 13, 9, 7, 11, 17 };

I will give the starting and ending point to my 'algorithm,

→ Selection_Sort (A , A + 10);

Now, we have entered the Algorithm's main function, let's sort it according to this Algorithm.

- Note : Our 1st Loop will seperate the sorted and Unsorted portion of Array, and 2nd one swaps the minimum element from Unsorted portion and manages at sorted portion.

1). Phase 1

```
15   2   8   4   1   13   9   7   11   17
↑
min,x
```

First for loop's initialization pointed 'x' and 'min' to the 1st element.

i).

```
15      2      8      4      1      13      9      7      11      17
↑       ↑
min, x  y
```

Now 2nd Loop will iterate the y variable and will also

- Now from Phase # 2 onwards, I'm skipping the Minimum Element finding checks.

- Phase 2

(No Swapping)

```
1    2    8    4    15    13    9    7    11    17
     ↑
     x, min
```

- Phase 3

```
          ┌ > ┐ ✓
1    2    8    4    15    13    9    7    11    17
          ↑    ↑
          x    min

1    2    4    8    15    13    9    7    11    17
```

- Phase 4

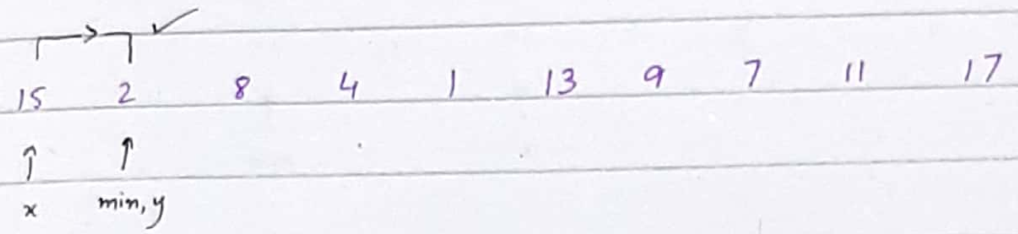
```
               ┌───────── > ─────────┐ ✓
1    2    4    8    15    13    9    7    11    17
               ↑                     ↑
               x                     min
```

- Phase 5

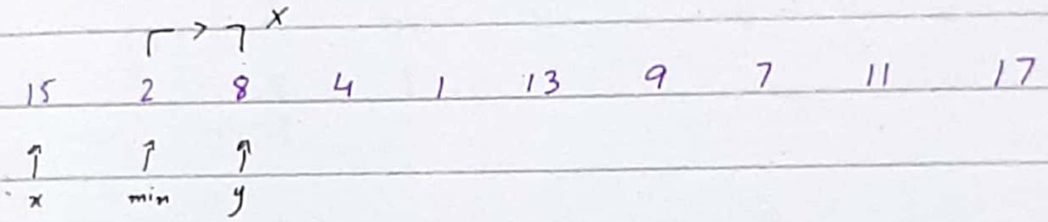
```
                    ┌──── > ────┐ ✓
1    2    4    7    15    13    9    8    11    17
                    ↑              ↑
                    x              min
```

initialize the 'min' at ~~every~~ first iteration. Now we will iterate the 'y' to the whole array to search for the minimum element.

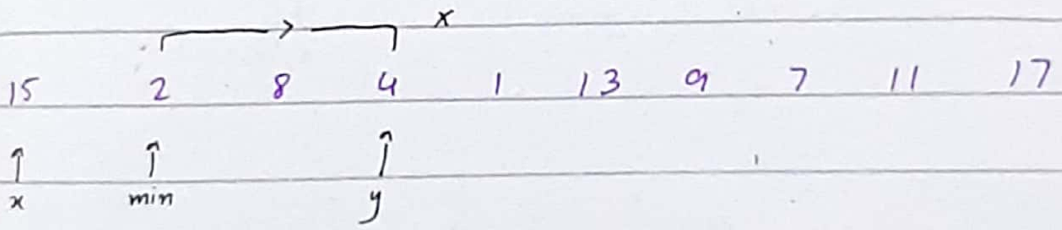
ii) →



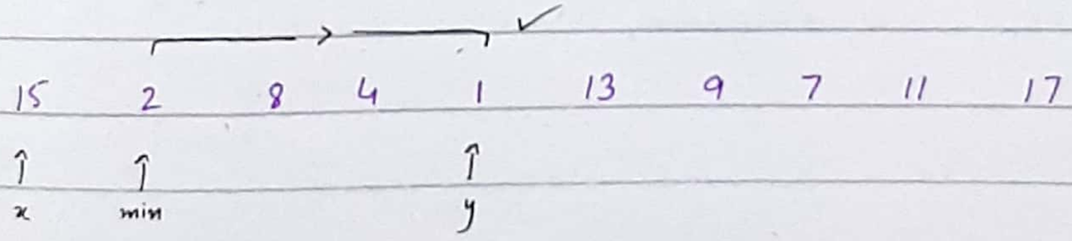
iii →



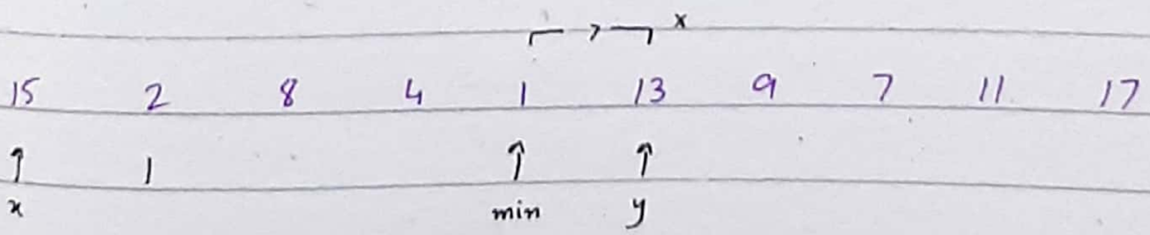
iv →



v →



vi →



Phase 6

$$1 \quad 2 \quad 4 \quad 7 \quad 8 \quad \overbrace{\underset{\underset{x}{\uparrow}}{13} \quad \underset{\underset{min}{\uparrow}}{9}}^{> \ \checkmark} \quad 15 \quad 11 \quad 17$$

Phase 7

$$1 \quad 2 \quad 4 \quad 7 \quad 8 \quad 9 \quad \overbrace{\underset{\underset{x}{\uparrow}}{13} \quad 15 \quad \underset{\underset{min}{\uparrow}}{11}}^{> \ \checkmark} \quad 17$$

Phase 8

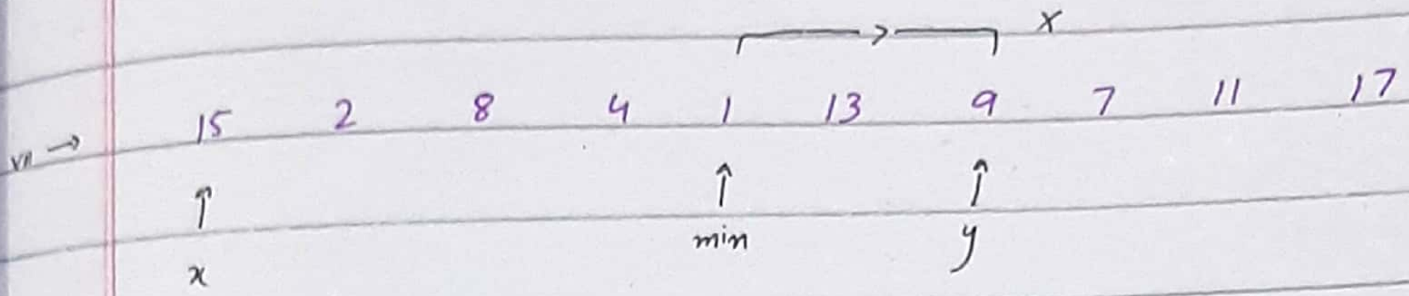
$$1 \quad 2 \quad 4 \quad 7 \quad 8 \quad 9 \quad 11 \quad \overbrace{\underset{\underset{x}{\uparrow}}{15} \quad \underset{\underset{y, min}{\uparrow}}{13}}^{>} \quad 17$$

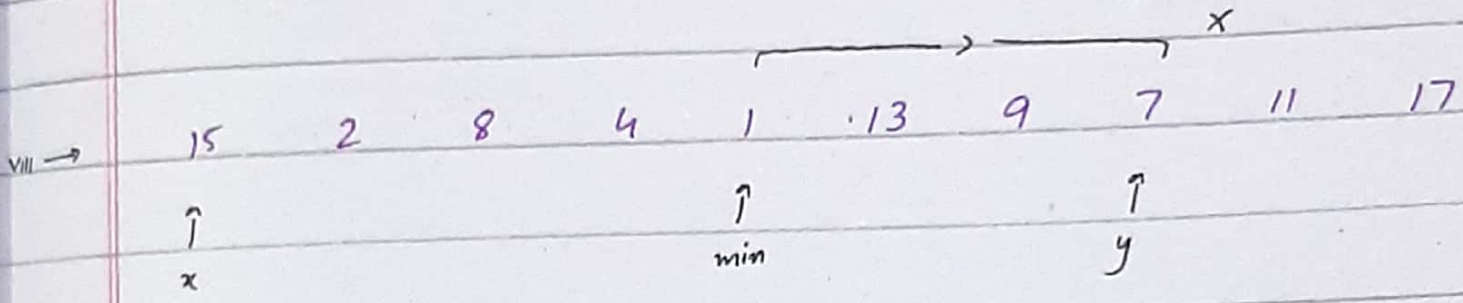
Phase 9

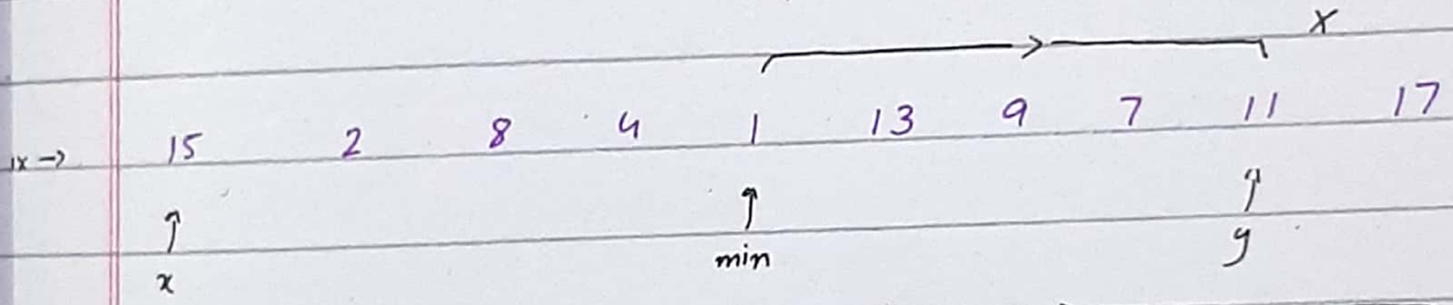
$$1 \quad 2 \quad 4 \quad 7 \quad 8 \quad 9 \quad 11 \quad 13 \quad \underset{\underset{x, min}{\uparrow}}{15} \quad 17$$

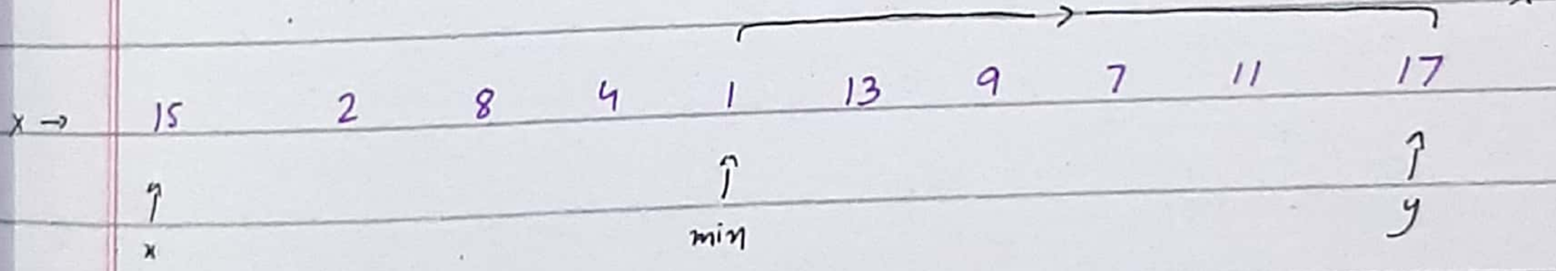
(No Swapping)

- Hence, we have successfully sorted the Given Array using Selection Sort Algorithm.

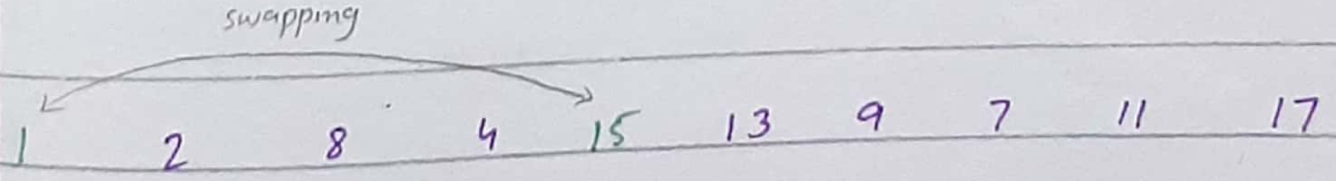








Now the Minimum Element will get exchanged by 'x'



b). $F(N) = O(G(N))$

Since, $f(N) = 2N^3 + N^2 + 2$

$g(N) = N^3$

for $f(N)$ to be $O[g(N)]$

$\Rightarrow f(N) \leq c.\ g(N)$

$\Rightarrow 2N^3 + N^2 + 2 \leq c.\ N^3$

~~forting~~, putting, $N = 1$,

$2 + 1 + 2 \leq c$

$c \geq 5 \quad \Rightarrow \quad c = 5$

$\Rightarrow 2N^3 + N^2 + 2 \leq 5N^3, \forall N \geq 1$

Verification

- for $N = 2$

  $22 \leq 40$ ✓

- for $N = 5$

  $277 \leq 625$ ✓

Hence, By Definition,

$2N^3 + N^2 + 2$ is $O(N^3)$

$\Rightarrow f(N)$ is $O(g(N))$ proved ✓

c). $f(N) = \Omega[H(N)]$

Since, $f(N) = 2N^3 + N^2 + 2$

$H(N) = 75$

for $f(N)$ to be $\Omega[H(N)]$ or $\Omega(1)$

$2N^3 + N^2 + 2 \geq c.1$ ~~75~~

$\therefore \Omega(75)$ can be written as $\Omega(1)$, as also $\Omega(75)$ represents constant rate growth.

for c, as we know that, the co-efficient of higest degree var. can be represented as, $c = 2$.

$\Rightarrow 2N^3 + N^2 + 2 \geq 2 \quad \forall N \geq 0$

Verification

- for $N = 2$

  $22 \geq 2$ ✓

- for $N = 5$

  $277 \geq 2$ ✓

Hence, By Definition,

$2N^3 + N^2 + 2$ is $\Omega(1)$

or, $f(N)$ is $\Omega[H(N)]$

a). $f(N) = \theta[G(N)]$

Since, $f(N) = 2N^3 + N^2 + 2$

$g(N) = N^3$

for $f(N)$ to be $\theta[g(N)]$

$O(N^3) \geq f(N) \geq \Omega(N^3)$

In Q#(b) we have proved that $f(N) \leq O(N^3)$, now for proving $f(N) \geq \Omega[g(N)]$

$\Rightarrow 2N^3 + N^2 + 2 \geq c. N^3$

As we know that, for $f(N)$ to be upper-bound on N^3, c should ~~have~~ be the co-efficient of Highest Degree Term i.e. $c = 2$

$\Rightarrow 2N^3 + N^2 + 2 \geq 2N^3$

As at, due to lower order terms, $f(N)$ will always be greater than $2N^3$, $\forall N \geq 0$

$\Rightarrow 2N^3 + N^2 + 2$ is $\Omega(N^3)$, By Definition

Hence,

$O(N^3) \geq 2N^3 + N^2 + 2 \geq \Omega(N^3)$

By Definition, $f(N)$ is $\theta(g(N))$

- Given That ;

for, $f(N) = 5N^2 + 2N$

$g(N) = N^2$,

$f(N)$ is $\Theta[g(N)]$

Hence, it implies that,

$O[g(N)] \geq f(N) \geq \Omega[g(N)]$

a). For, $f(N)$ is $O[g(N)]$,

~~if~~ for $f(N)$ ~~is~~ to be $O[g(N)]$

$f(N) \leq c.\ g(N)$, upper bound case.

Now, for 'c', putting $N = 1$,

$5N^2 + 2N \leq c.\ N^2$

$5 + 2 \leq c \Rightarrow c \geq 7$, $\forall\ n \geq 1$

Hence, $5N^2 + 2N \leq 7N^2$

Verification

for $N = 2$

$20 + 4 \leq 28$ ✓

for $N = 5$

$125 + 10 \leq 175$ ✓

Hence, $7N^2$ is upper bound on $5N^2 + 2N$

$\Rightarrow 5N^2 + 2N \leq c.\ N^2$

$\Rightarrow f(N)$ is $O[g(N)]$ proved.

b). $F(N)$ is $\Omega\,[g(N)]$

for $f(N)$ to be $\Omega\,[g(N)]$

$f(N) \geq c.\, g(N)$

$\Rightarrow 5N^2 + 2N \geq c.\, N^2$

As we know that, in case of Ω or lower bound, the co-efficient of Highest degree term of $f(N)$ is 'c'. i.e. $c = 5$

$\Rightarrow 5N^2 + 2N \geq 5N^2 \quad , \forall N \geq 0$

Verification

for $N = 0$ | for $N = 5$

$0 \geq 0$ ✓ | $125 + 10 \geq 125$ ✓

Hence, $5N^2$ is Lower Bound of $f(N)$

$\Rightarrow 5N^2 + 2N \geq c.\, N^2$

$\Rightarrow f(N)$ is $\Omega\,[g(N)]$ proved ✓

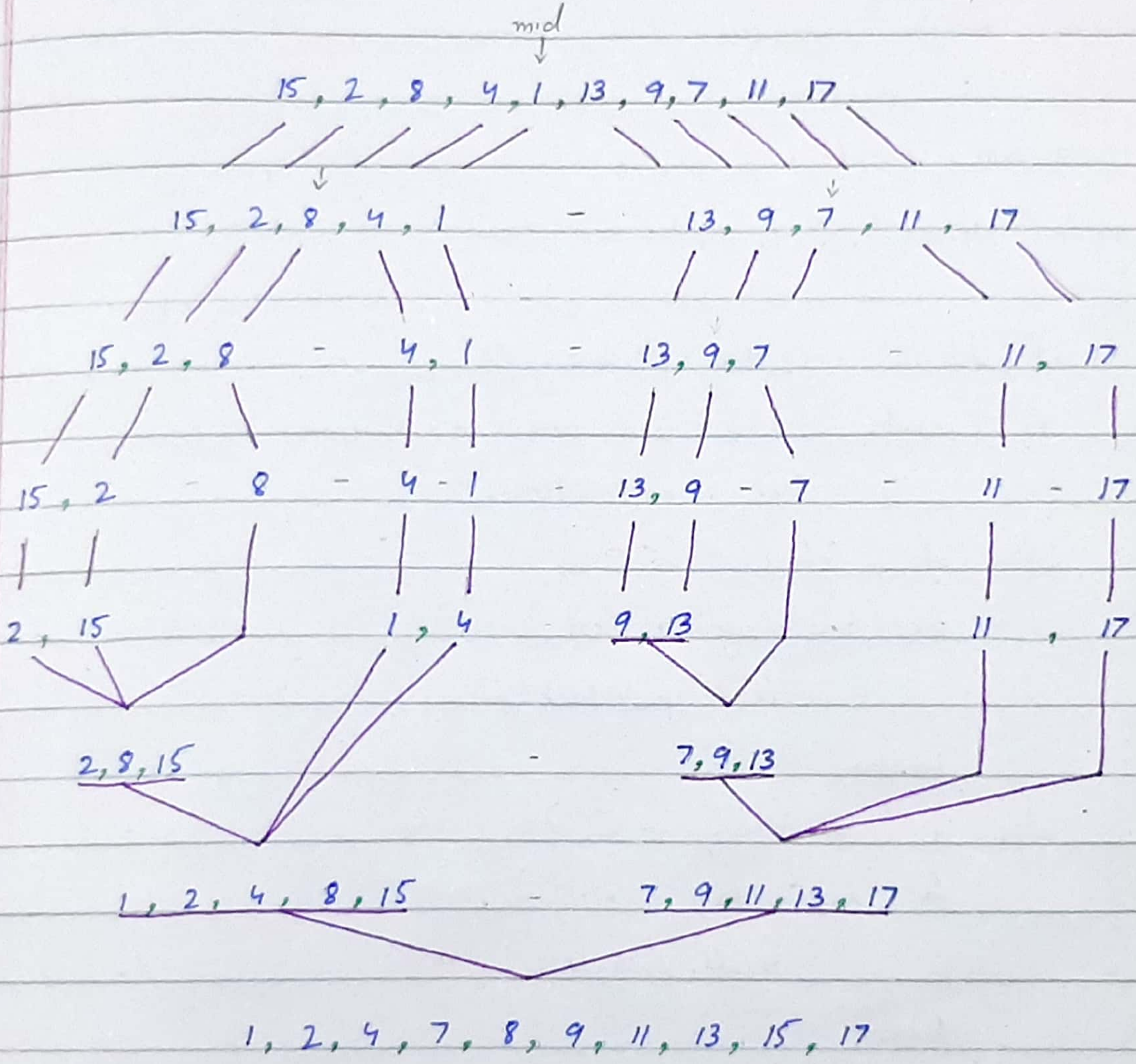
Modern C++ Implementation

```
vector<int> mergeSort(vector<int>& m)
{
    if (m.size() < 2) return m;                                    1
    vector<int> left, right;                                       2
    int middle = (m.size()+1)/2;                                   3
    left.reserve(middle), right.reserve(m.size());                 4

    for (int i=0; i < middle; ++i)                                 5
            left.emplace_back(m[i]);                               6


    for (int i = middle; i < m.size(); ++i)                        7
            right.emplace_back(m[i]);                              8


    return merge(mergeSort(left), mergeSort(right));
}
                                                                   9
```

Modern C++ Implementation of Merge()

```
vector<int> merge (vector<int> left, vector<int> right )
{
    vector<int> result ;                                        1
    while ( left.size()>0 || right.size() > 0 )                 2
    {
        if ( left.size()>0 && right.size()>0 )                  3
            result. push_back ( (left. front()<= right. front() ?
                    left  :  right ). front() );                4
        else  if ( left.size() > 0 )                            5
            for (auto& elem : left )                            6
                    result. push. back (elem) ;                 7
            break ;
        else  if ( right. size () > 0 )                         8
            for ( auto& elem : right )                          9
                    result. push_back (elem ) ;                 10
            break ;
    }
    return result ;                                             11
}
```

- General Working Mechanism



- Detailed Method of Sorting via Merge Sort

- Given_Array = { 15, 2, 8, 4, 1, 13, 9, 7, 11, 17 };
- To Sort this Array, lets pass it to mergeSort() and visualize its execution.

mergeSort (Given_Array);

- Division Step // for { Given_Array }

Now, according to the instructions, from (2 to 8) in mergeSort function, the Given Array will be divided into two parts; left and right, according to the mid-point of Given-Array, which is index #4 = 1.

left = { 15, 2, 8, 4, 1 };
right = { 13, 9, 7, 11, 17 };

- Recurrence // for left

At line # 9 of mergeSort(), according to the order of execution, the mergeSort() will again be called for the 'left' of Given Array, which is ;

left = { 15, 2, 8, 4, 1 };
which is now = m ;

Division of Left // for {15, 2, 8, 4, 1 }

Now again, line # ;
2 will initialize two vectors of left and right for 'm'.
3 will calculate the middle point, which is '2'.
4 will allocate the required space in left and right.
from 5 - 6, will fill up the left side of m, which is
left = { 15, 2, 8 };
from 7 - 8, will fill up the Right side of m, which is
right = { 4, 1 };

Recurrence // for left {15, 2, 8 }

Now again according to the order of execution, 'left' will be passed to merge Sort() for further division, which is : 'left = { 15, 2, 8 }

- **Division** // for Left {15, 2, 8}

Now again, instructions from 2 to 8 will divide the 'm' vectror into two halves, left and right,

left = { 15, 2 }; ∴ middle point = 2

right = { 8 };

- **Recurrence** // for left

Again, according to the order of execution, left = { 15, 2 } will be passed to mergeSort().

- **Division** // for { 15, 2 } left

Now, m = { 15, 2 } will be divided into left and right, which will have similar number of elements.

left = { 15, }

right = { 2 }

- **Recurrence** // for { 15 } left

Instruction at line # 9 will execute mergeSort ()

for : left = { 15 }

- **Base Step** // for {15} - left

Now, in mergeSor(). line # 1. According to this condition, as m.size() is '1', m will be returned to line # 9 of previous Stack-frame.

- **Recurrence** and Base Case for Right

Now, according to the order of execution, right = {2} will be passed to mergeSort() and just like before, it'll return {2} to current stack frame.
Now, line #9 becomes ;

merge ({15}, {2});

Since, we've reached the Base Case, now merge operation of these elements will be performed.

- **Merge** // for {15} and {2}

In merge(), conditions at (2) and (3) will return true, and according to the instruction at (4), {2} will be inserted, before {15}, in result vector initialized at (1). Now, this merg() will return ~~{15, 2}~~ {2, 15} to previous stack frame, which will become,

merge ({2, 15}, {8}); ∴ Single Element is returned, when size is < 2.

Now, merge() for {2,15} and {8} will be called.

- Merge // for {2, 15}, {8}

Now, due to the satisfection of conditions at (2) & (3), and according to line (4), {2} will be inserted in the result vector. Now, due to dissatisfaction of condition at (4), {8} will be inserted in result vector.

Now, since, right became empty, condition at (3) will not satisfy, and according to condition at (5), this for_each loop will insert {15} to the result, which will become ;

result : {2, 8, 15};

Note :-

As we can see, merge() recieves two vectors and orderly merge the resultant vector which is finally returned.

- Recurrence

Now, this resultant vector from merge() is returned to the previous stack frame and it'll become,

merge ({2, 8, 15 }, mergeSort ({4, 1 }));

Now, after the Returning of sorted elements from mergeSort ({4, 1 }), the current Stack frame will get :

merge { {2, 8, 15 }, {1, 4 });

- Merge

Now, after the satisfaction of conditions at (2) and (3), three times, result will have :

result = { 1, 2, 4 } ;

Now, Since, right = {} is now empty, so Code-Block from 5 to 7 will fill the result with 'left' elements

result = { 1, 2, 4, 8, 15 } ;

Now, it'll be returned to previous stack-frame.

- Recurrence

Now, the result from previous stack frame will be returned to first stack frame, which is :

merge({1,2,4,8,15}, mergeSort({13,9,7,11,17}) ;

Now, Similar steps will be repeated for mergeSort({13, 9, 7, 11, 17}); and finally we'll have,

merge({1, 2, 4, 8, 15}, {7, 9, 11, 13, 17});

- Merge // final call

This time, condition at (2) and (3) will satisfy in 9/10 iterations, and due to instruction at (4), we will have :

result = {1, 2, 4, 7, 8, 9, 11, 13, 15} ;

Since, at 10th iteration, left vector = {} will be completely empty, and condition at (8) will satisfy instead of, at (3) and (5), and for loop will insert final element to the Result.

result = {1, 2, 4, 7, 8, 9, 11, 13, 15, 17} ✓

Q5 a). Array : { 18, 22, 20, 25, 30, 44, 60, 51, 37 }

Note : Since, Binary Search is never be applied on un-sorted array, and the given array is unsorted. So, I personally have solved this Question Two Times with Sorted and unsorted manner.

Binary Search on Un-Sorted Given Array :

Array [18, 22, 20, 25, 30, 44, 60, 51, 37] , size = 9

18	22	20	25	30	44	60	51	37
0	1	2	3	4	5	6	7	8

=> low = 0 , high = size-1 = 8 , value = 60

1st Iteration

→ low ≤ high (i.e. 0 ≤ 8) ✓

↳ mid = (Low+high) / 2 = 4

↳ (A[4] = 30) < (value = 60) ✓

↳ Low = mid + 1 = 4+1 = 5

IInd Iteration

→ low ≤ high (i.e 5 ≤ 8) ✓

↳ mid = (5+8)/2 = 6

↳ (A[6] = 60) => (value = 60) ✓

↳ return mid = 60

Binary Search on Sorted Array

Array [18, 20, 22, 25, 30, 37, 44, 51, 60] , Size = 9

=> Low = 0, high = size - 1 = 8 , value = 60

1st Iteration

→ low ≤ high (0 ≤ 8) ✓

↳ mid = $\frac{\text{Low+high}}{2}$ = 4

↳ A[4] = 30 < value ✓

↳ low = mid+1 = 4+1 = $\underline{5}$

2nd Iteration

→ Low ≤ high (5 ≤ 8) ✓

↳ mid = $\frac{\text{Low + high}}{2}$ = $\frac{5+8}{2}$ = $\underline{6}$

↳ A[6] = 44 < value ✓

↳ Low = mid+1 = 6.

3rd Iteration

→ Low ≤ high (6 ≤ 8) ✓

↳ mid = $\frac{\text{Low + high}}{2}$ = 7

↳ A[7] = 51 < value ✓

↳ Low = mid + 1 = $\underline{8}$

4th Iteration

→ Low ≤ high (8 ≤ 8) ✓

↳ mid = $\frac{\text{Low + high}}{2}$ = 8

↳ A[8] = 6 == value ✓

→ return mid = 8

• Main Code	Time Units	Frequency
int q = 0;	1	1
x= func (a,b,c);	1	1
if (x>=0)	1	1
for (int i=0; i<=n; ++i)	1, 1, 1	$1 + (n+2) + (n+1)$
cout << "..." << i;	1	$n+1$
else		-
for (int j=n; j>0; j/=4)	1, 1, 1	$1 + \log_4(n) + 1 + \log_4(n$
cout << "..." <<j<<;	1	$\log_4(n)$
for (int m=1; m<n; ++m)	1, 1, 1	$1, n + n-1$
q += m;	2	$n-1$
cout<< "..." << m << "..." <<q;	1	$n-1$
for(int k=1; k<=n; ++k)	1, 1, 1	$1 + n+1 + n$
// Something	1	n
for(int p=0; p<n; ++p)	1, 1, 1	$n(1 + n+1 + n)$
// Something	1	$(n)(n)$
for (int a=0; a<=n; ++a)	1, 1, 1	$n^2(1 + n+2 + n+1)$
// Something	1	n^2

• Note : In If-Else of line 3, Time Complexity of if will be summed withe the Total as it has higher T(N) than else.

• Total (N) = $3 + (2n+4) + (n+1) + (2n) + (3n-3) + (2n+2) + (n) + (2n^2 + 2n) + n^2 + (2n^3 + 4n^2) + n^3 + n^2 + 1$

=> T(N) = $3n^3 + 8n^2 + 13n + 8$