DATA STRUCTURE NOTES

Searching and sorting in C

Written by:
SAMUNDAR SINGH

Searching Technique

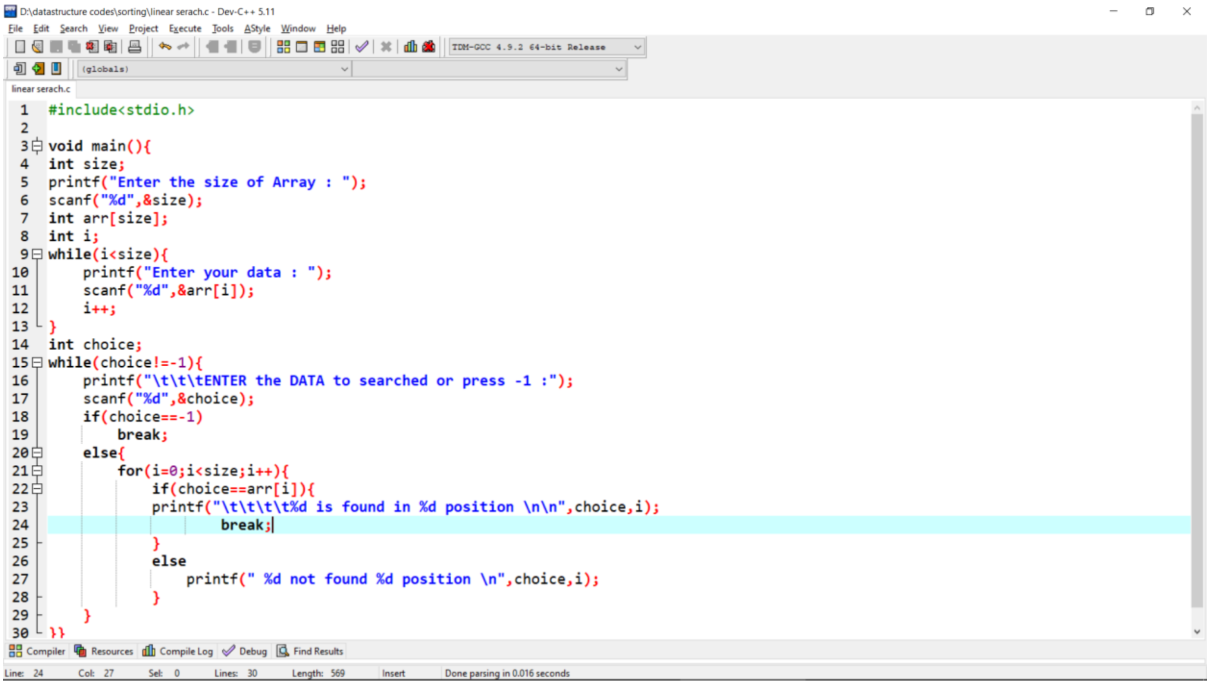
Linear search:

it is a process of searching the data in an array for a particular value. It works by comparing the value to be searched with every element of the array one by one until match is found.

Time complexity:

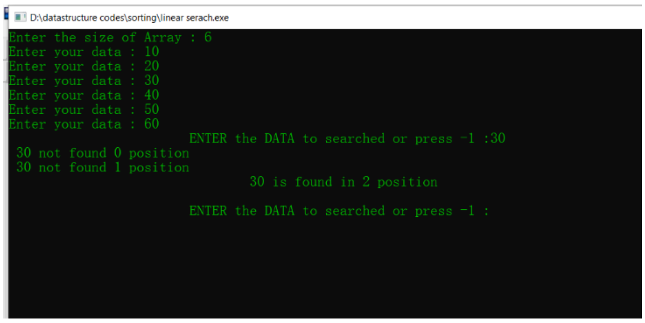
Case	Best case	Worst case	Average case
If item is present	**1**	**n**	**n/2**
If item is not present	**n**	**n**	**n**

Code to implement the linear search:


D:\datastructure codes\sorting\linear serach.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
linear serach.c
Compiler Resources Compile Log Debug Find Results
Line: 24 Col: 27 Sel: 0 Lines: 30 Length: 569 Insert Done parsing in 0.016 seconds

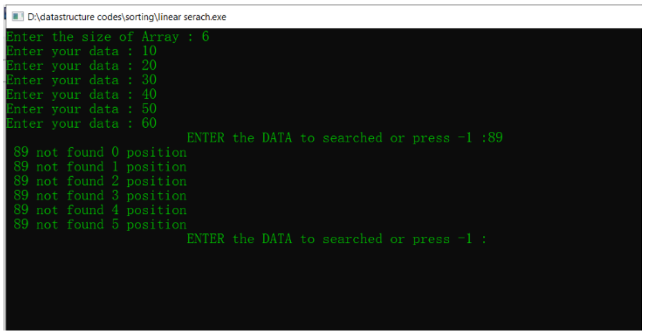
```c
#include<stdio.h>

void main(){
int size;
printf("Enter the size of Array : ");
scanf("%d",&size);
int arr[size];
int i;
while(i<size){
    printf("Enter your data : ");
    scanf("%d",&arr[i]);
    i++;
}
int choice;
while(choice!=-1){
    printf("\t\t\tENTER the DATA to searched or press -1 :");
    scanf("%d",&choice);
    if(choice==-1)
        break;
    else{
        for(i=0;i<size;i++){
            if(choice==arr[i]){
            printf("\t\t\t\t%d is found in %d position \n\n",choice,i);
                    break;
            }
            else
                printf(" %d not found %d position \n",choice,i);
            }
    }
}}
```

When data is found then no need to execute the whole loop



```
Enter the size of Array : 6
Enter your data : 10
Enter your data : 20
Enter your data : 30
Enter your data : 40
Enter your data : 50
Enter your data : 60
                        ENTER the DATA to searched or press -1 :30
 30 not found 0 position
 30 not found 1 position
                                30 is found in 2 position

                        ENTER the DATA to searched or press -1 :
```

When data is not found then although we have to execute the whole loop.



```
Enter the size of Array : 6
Enter your data : 10
Enter your data : 20
Enter your data : 30
Enter your data : 40
Enter your data : 50
Enter your data : 60
                        ENTER the DATA to searched or press -1 :89
 89 not found 0 position
 89 not found 1 position
 89 not found 2 position
 89 not found 3 position
 89 not found 4 position
 89 not found 5 position
                        ENTER the DATA to searched or press -1 :
```

DATA STRUCTURE NOTES

Binary search:

- To initiate binary search first we have to arrange the data in an ordered manner (either increasing or decreasing manner).
- This search starts from the middle term of array.

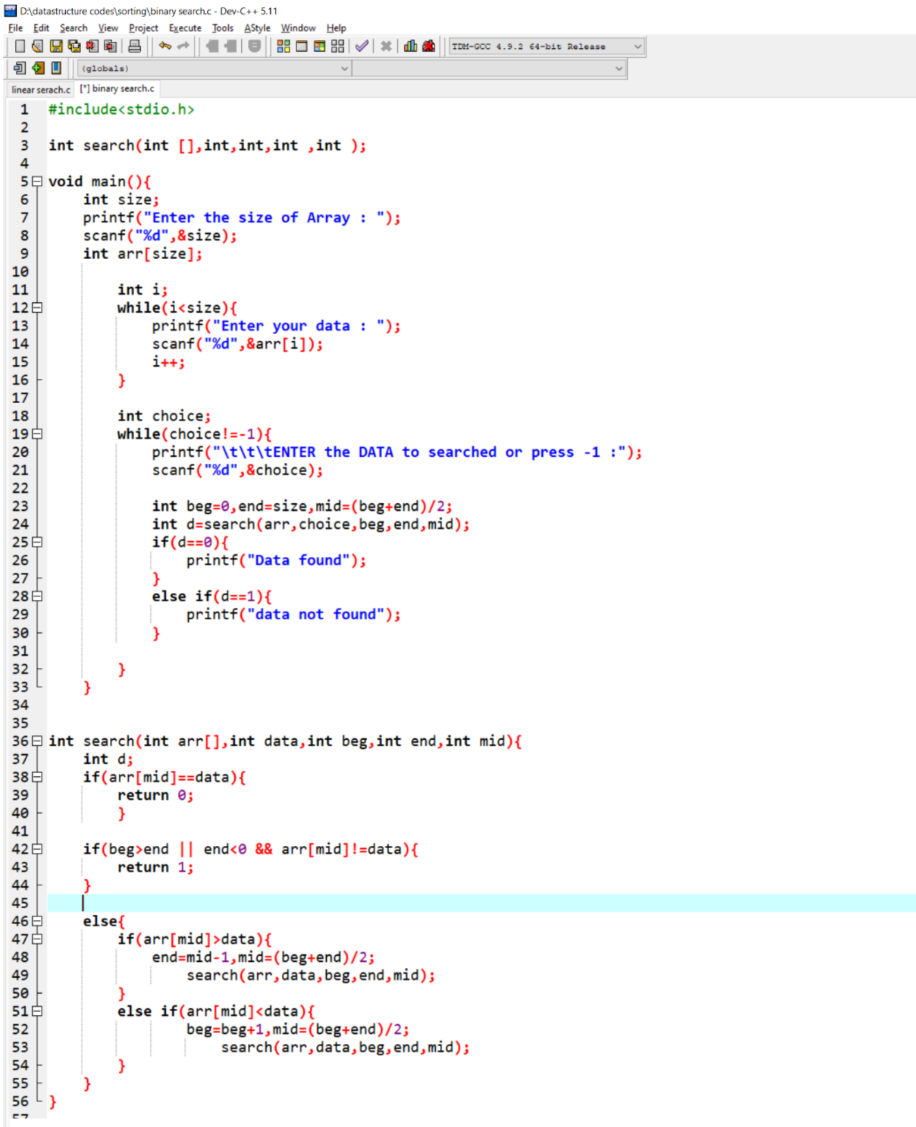
Logic:

1. If the **arr[middle term] == target value** then it simply returns true.
2. If our **target_item >arr[middle item]** the we have to search it in the right section and forget the left section. By doing so once we know that the data in the right section of array then we have to change the value of beginning and middle, end will remain the same.

   beg= mid+1 and mid= (beg +end)/2
3. If our **target_item<arr[middle item]** then we have to search in the left section and forget the left section. By doing so once we know that the data in the right section of array then we have to change the value of end and middle, beg will remain the same.

   end= mid-1 and mid= (beg +end)/2

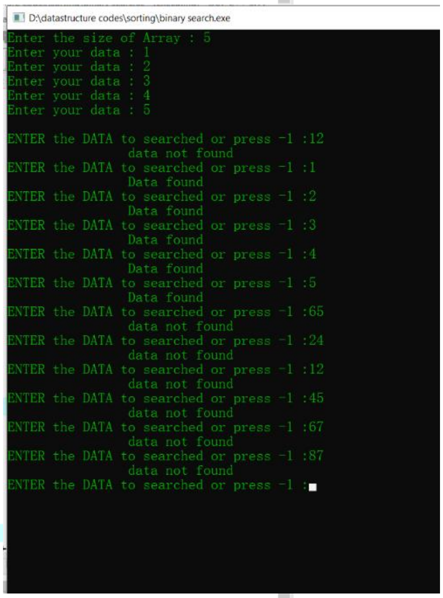
Code to implement the linear search:



D:\datastructure codes\sorting\binary search.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
linear serach.c
[*] binary search.c

```c
#include<stdio.h>

int search(int [],int,int,int ,int );

void main(){
	int size;
	printf("Enter the size of Array : ");
	scanf("%d",&size);
	int arr[size];
	
		int i;
		while(i<size){
			printf("Enter your data : ");
			scanf("%d",&arr[i]);
			i++;
		}
		
		int choice;
		while(choice!=-1){
			printf("\t\t\tENTER the DATA to searched or press -1 :");
			scanf("%d",&choice);
			
			int beg=0,end=size,mid=(beg+end)/2;
			int d=search(arr,choice,beg,end,mid);
			if(d==0){
				printf("Data found");
			}
			else if(d==1){
				printf("data not found");
			}
			
		}
	}


int search(int arr[],int data,int beg,int end,int mid){
	int d;
	if(arr[mid]==data){
		return 0;
		}
		
	if(beg>end || end<0 && arr[mid]!=data){
		return 1;
	}
	
	else{
		if(arr[mid]>data){
			end=mid-1,mid=(beg+end)/2;
				search(arr,data,beg,end,mid);
		}
		else if(arr[mid]<data){
				beg=beg+1,mid=(beg+end)/2;
					search(arr,data,beg,end,mid);
		}
	}
}
```

Output:



```
Enter the size of Array : 5
Enter your data : 1
Enter your data : 2
Enter your data : 3
Enter your data : 4
Enter your data : 5

ENTER the DATA to searched or press -1 :12
                        data not found
ENTER the DATA to searched or press -1 :1
                        Data found
ENTER the DATA to searched or press -1 :2
                        Data found
ENTER the DATA to searched or press -1 :3
                        Data found
ENTER the DATA to searched or press -1 :4
                        Data found
ENTER the DATA to searched or press -1 :5
                        Data found
ENTER the DATA to searched or press -1 :65
                        data not found
ENTER the DATA to searched or press -1 :24
                        data not found
ENTER the DATA to searched or press -1 :12
                        data not found
ENTER the DATA to searched or press -1 :45
                        data not found
ENTER the DATA to searched or press -1 :67
                        data not found
ENTER the DATA to searched or press -1 :87
                        data not found
ENTER the DATA to searched or press -1 :
```

Time complexity

Case	Best Case	Worst Case	Average Case
If item is present	1	O(log n)	O(log n)
If item is not present	O(log n)	O(log n)	O(log n)

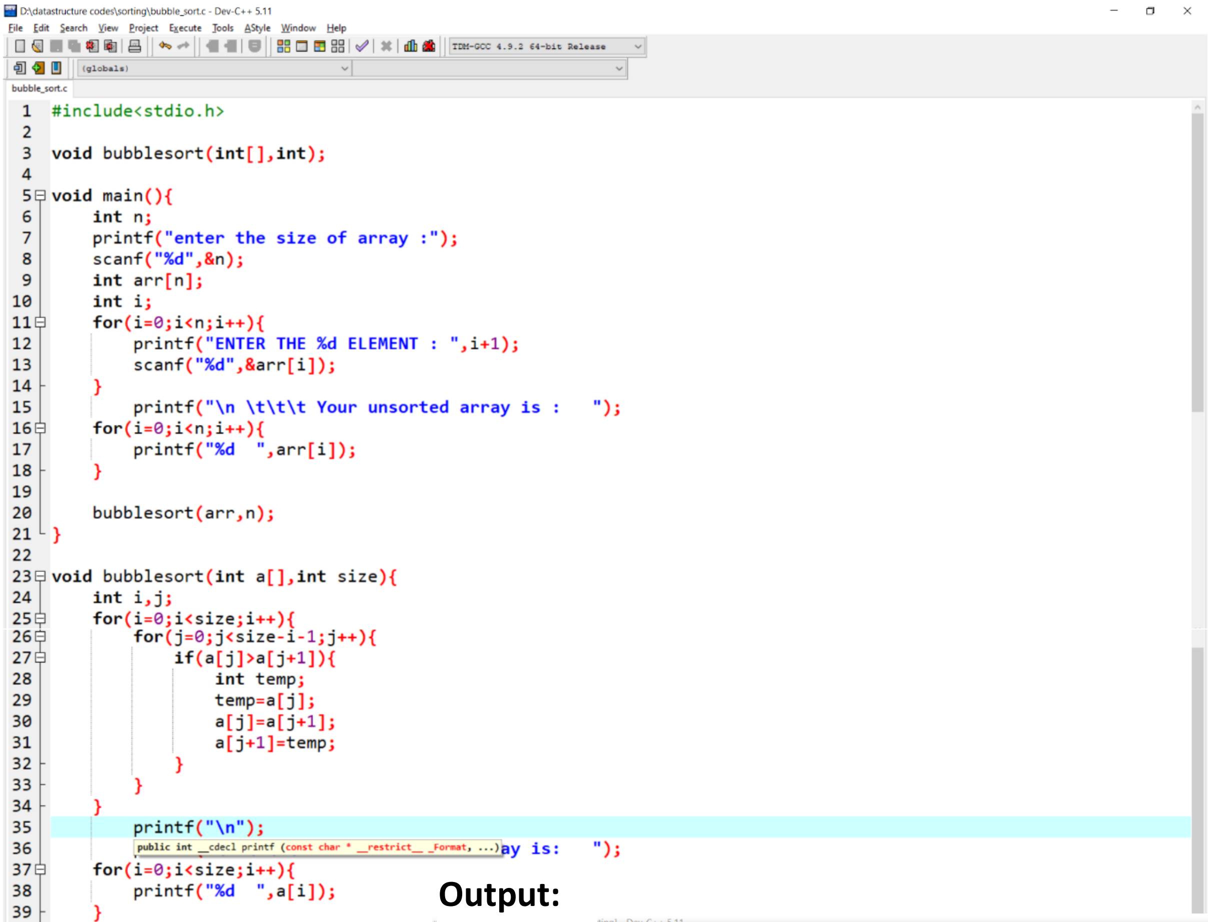
Sorting Technique

Bubble Sort:

- In bubble sort consecutive adjacent pair of elements in the array are compared with each other using loops (only two loops are required to perform bubble sort).
- It simply put one element from start of array and make comparison to all the element of the array if it is largest from all element then it places it to the last position (for ascending order).
- It means that during first pass the largest value is placed on the last index of array and in second pass the second largest value place in the second largest index of array and this process continues until all the element is not sorted.
- Now why we call it bubble sort? Because the highest value raises like a bubble from array and put on its designated place.

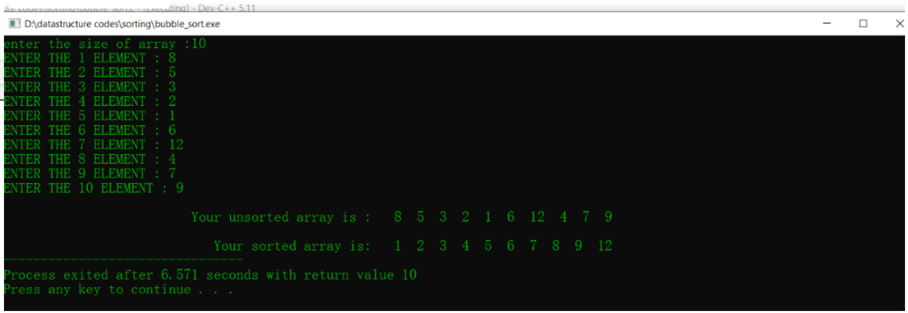
Time complexity: O (n^2) .

Program to implement Bubble sort:


D:\datastructure codes\sorting\bubble_sort.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
bubble_sort.c
public int __cdecl printf (const char * __restrict__ _Format, ...)
Compiler Resources Compile Log Debug Find Results
Line: 35 Col: 19 Sel: 0 Lines: 41 Length: 702 Insert

```
#include<stdio.h>

void bubblesort(int[],int);

void main(){
    int n;
    printf("enter the size of array :");
    scanf("%d",&n);
    int arr[n];
    int i;
    for(i=0;i<n;i++){
        printf("ENTER THE %d ELEMENT : ",i+1);
        scanf("%d",&arr[i]);
    }
        printf("\n \t\t\t Your unsorted array is :   ");
    for(i=0;i<n;i++){
        printf("%d  ",arr[i]);
    }

    bubblesort(arr,n);
}

void bubblesort(int a[],int size){
    int i,j;
    for(i=0;i<size;i++){
        for(j=0;j<size-i-1;j++){
            if(a[j]>a[j+1]){
                int temp;
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
        printf("\n");
        public int __cdecl printf (const char * __restrict__ _Format, ...)ay is:   ");
    for(i=0;i<size;i++){
        printf("%d  ",a[i]);
    }
}
```

Output:



```
enter the size of array :10
ENTER THE 1 ELEMENT : 8
ENTER THE 2 ELEMENT : 5
ENTER THE 3 ELEMENT : 3
ENTER THE 4 ELEMENT : 2
ENTER THE 5 ELEMENT : 1
ENTER THE 6 ELEMENT : 6
ENTER THE 7 ELEMENT : 12
ENTER THE 8 ELEMENT : 4
ENTER THE 9 ELEMENT : 7
ENTER THE 10 ELEMENT : 9

                         Your unsorted array is :   8  5  3  2  1  6  12  4  7  9

                            Your sorted array is:   1  2  3  4  5  6  7  8  9  12

Process exited after 6.571 seconds with return value 10
Press any key to continue . . .
```

For better understanding we take an example and see the flow of execution:

Let us consider an (unsorted) array arr = [8, 5, 3, 1, 2, 6] here n=6(size of array)

Pass 1: **our array is arr = [8, 5, 3, 1, 2, 6]**

i = 0 and (j =0 ; j < n-i-1; j++) it means that loop in j will execute up to j <n – i -1➔ j < 5

j=0	arr[0] > arr[1]	true	swapping done	[5 ,8, 3, 1, 2, 6]
j=1	arr[1] > arr[2]	true	swapping done	[5 ,3 ,8, 1, 2, 6]
j=2	arr[2] > arr[3]	true	swapping done	[5 ,3 ,1, 8, 2, 6]
j=3	arr[3] > arr[4]	true	swapping done	[5 ,3 , 1, 2, 8, 6]
j=4	arr[1] > arr[2]	true	swapping done	[5 ,3 , 1, 2, 6, 8]

in first pass the highest element is placed at the last index of array.

Pass 2: **our array is arr = [5 ,3 , 1, 2, 6, 8]**

i = 1 and (j =0 ; j < n-i-1; j++) it means that loop in j will execute up to j <n – i -1➔ j < 4

j=0	arr[0] > arr[1]	true	swapping done	[3, 5, 1, 2, 6, 8]
j=1	arr[1] > arr[2]	true	swapping done	[3, 1, 5, 2, 6, 8]
j=2	arr[2] > arr[3]	true	swapping done	[3, 1, 2, 5, 6, 8]
j=3	arr[3] > arr[4]	false	NOTHING	

in second pass the second highest element is placed at the second largest index of array,

Pass 3: **our array is arr = [3, 1, 2, 5, 6, 8]**

i = 2 and (j =0 ; j < n-i-1; j++) it means that loop in j will execute up to j <n – i -1➔ j < 3

j=0	arr[0] > arr[1]	true	swapping done	[1, 3, 2, 5, 6, 8]
j=1	arr[1] > arr[2]	true	swapping done	[1, 2, 3, 5, 6, 8]
j=2	arr[2] > arr[3]	false	NOTHING	

In third pass the third highest element is placed at the third largest index of array, and so on…….

Now although the array is sorted but still the loop will executed that is the big demerit of bubble sort

Pass 4: **our array is arr = [1, 2, 3, 5, 6, 8]**

i = 3 and (j =0 ; j < n-i-1; j++) it means that loop in j will execute up to j <n – i -1➔ j < 2

j=0	arr[0] > arr[1]	false	NOTHING
j=1	arr[1] > arr[2]	false	NOTHING

Pass 5: **our array is arr = [1, 2, 3, 5, 6, 8]**

i = 4 and (j =0 ; j < n-i-1; j++) it means that loop in j will execute up to j <n – i -1➔ j < 1

j=0	arr[0] > arr[1]	false	NOTHING

Sorted data

Pass 5: **our array is arr = [1, 2, 3, 5, 6, 8]**

i = 5 and (j =0 ; j < n-i-1; j++) it means that loop in j will execute up to j <n – i -1➔ j < 0

further no execution

DATA STRUCTURE NOTES

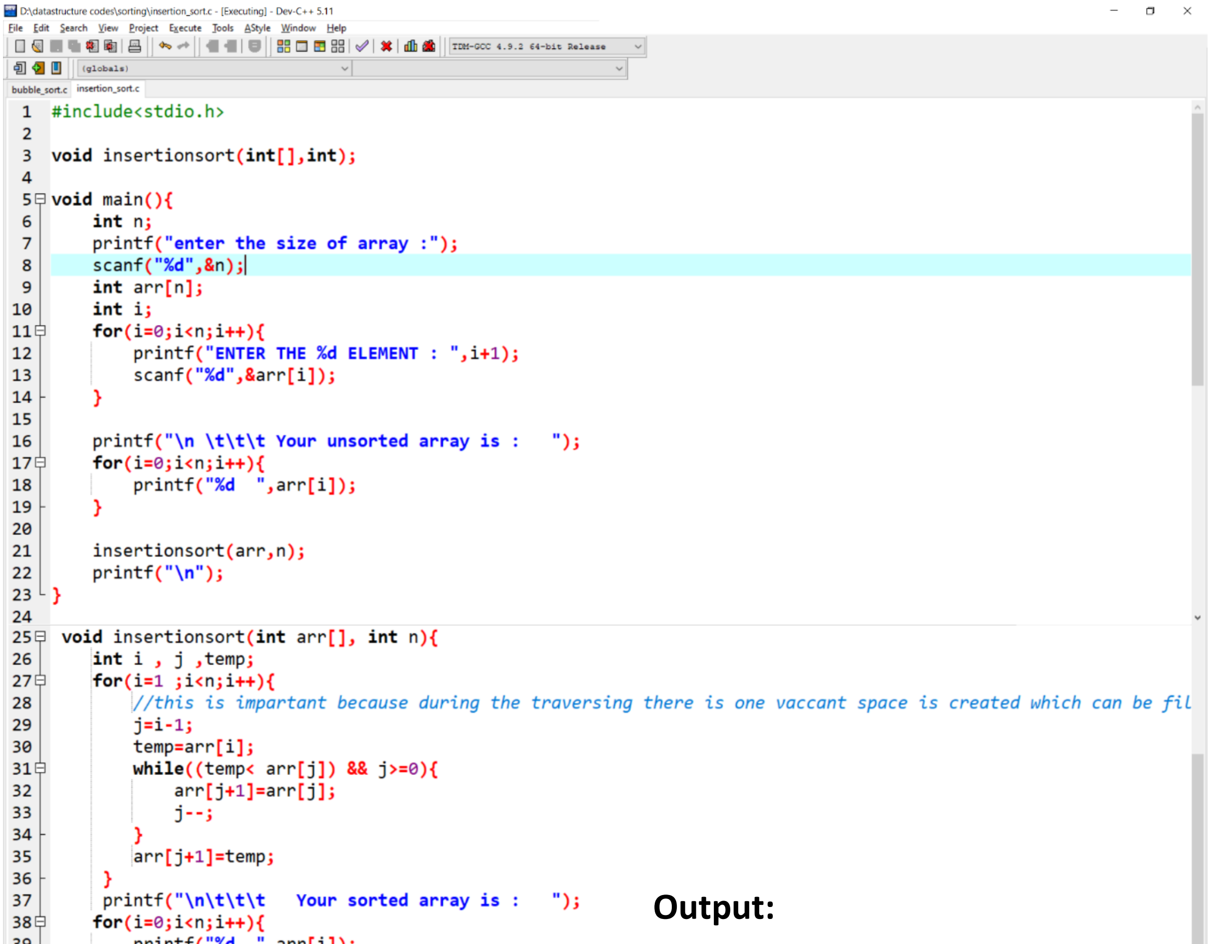
Insertion Sort:

- In insertion sort simply one data is picked up and at appropriate place it is inserted
- The array is divided into two set sorted and unsorted set , then we select a data and traversed the sorted set to insert it(we assumed that the first element is sorted).
- As we are moving from left to right one element has picked and then an appropriate place is searched after that we insert the data.
- There is no need to search the whole element only we have to searched from the sorted set due to which the time complexity quite reduced.

Time complexity:	
Best case : O (n)	worst case :O (n^2)

- When the array is already sorted then there is best case occur and when the data is in reverse order then the worst case occur.

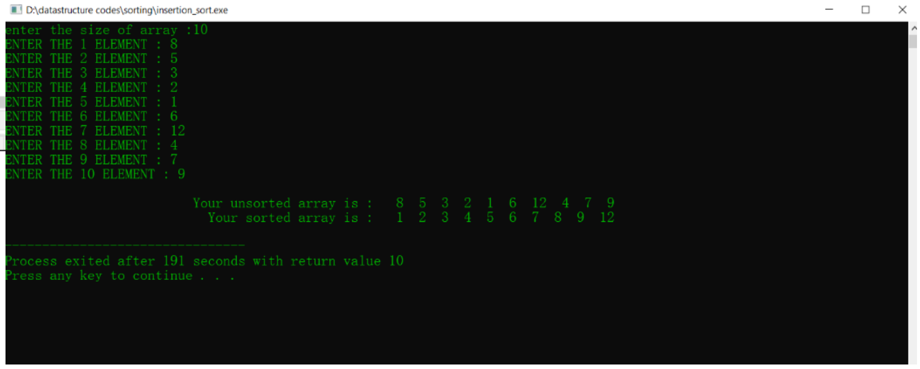
Program to implement insertion sort:



D:\datastructure codes\sorting\insertion_sort.c - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
bubble_sort.c
insertion_sort.c
Compiler
Resources
Compile Log
Debug
Find Results
Line: 22
Col: 18
Sel: 0
Lines: 42
Length: 838

```c
#include<stdio.h>

void insertionsort(int[],int);

void main(){
    int n;
    printf("enter the size of array :");
    scanf("%d",&n);
    int arr[n];
    int i;
    for(i=0;i<n;i++){
        printf("ENTER THE %d ELEMENT : ",i+1);
        scanf("%d",&arr[i]);
    }

    printf("\n \t\t\t Your unsorted array is :   ");
    for(i=0;i<n;i++){
        printf("%d  ",arr[i]);
    }

    insertionsort(arr,n);
    printf("\n");
}

 void insertionsort(int arr[], int n){
    int i , j ,temp;
    for(i=1 ;i<n;i++){
        //this is impartant because during the traversing there is one vaccant space is created which can be fil
        j=i-1;
        temp=arr[i];
        while((temp< arr[j]) && j>=0){
            arr[j+1]=arr[j];
            j--;
        }
        arr[j+1]=temp;
     }
     printf("\n\t\t\t   Your sorted array is :   ");
    for(i=0;i<n;i++){
        printf("%d  ",arr[i]);
    }

 }
```

Output:



```
enter the size of array :10
ENTER THE 1 ELEMENT : 8
ENTER THE 2 ELEMENT : 5
ENTER THE 3 ELEMENT : 3
ENTER THE 4 ELEMENT : 2
ENTER THE 5 ELEMENT : 1
ENTER THE 6 ELEMENT : 6
ENTER THE 7 ELEMENT : 12
ENTER THE 8 ELEMENT : 4
ENTER THE 9 ELEMENT : 7
ENTER THE 10 ELEMENT : 9

                         Your unsorted array is :   8  5  3  2  1  6  12  4  7  9
                           Your sorted array is :   1  2  3  4  5  6  7  8  9  12

--------------------------------
Process exited after 191 seconds with return value 10
Press any key to continue . . .
```

Written by:
SAMUNDAR SINGH

For better understanding we take an example and see the flow of execution:

Let us consider an (unsorted) array arr = [3, 8, 5, 2, 4, 1] here n=6(size of array)

Pass 1: **our array is arr = [3, 8, 5, 2, 4, 1]**

i = 1 temp=arr[1] = 8 j = i - 1= 0

j=0	temp<arr[0](8<0)	false	NOTHING	[3 , 8, 5, 2, 4, 1]

Pass 2: **our array is arr = [3, 8, 5, 2, 4, 1]**

i = 2 temp=arr[2] = 5 j = 2 - 1= 1

j=1	temp<arr[1](5<8)	true	data picked and shift	[3 , ○,8 , 2, 4, 1]
j=0	temp<arr[0](5<3)	false	data inserted	[3, 5, 8, 2, 4, 1]

Pass 3: **our array is arr = [3, 5, 8, 2, 4, 1]**

i = 3 temp=arr[3] =2 j = 3 - 1= 2

j=2	temp<arr[2](2<8)	true	data picked and shift	[3 ,5 ,○ ,8 , 4, 1]
j=1	temp<arr[1](2<5)	true	space shift	[3 , ○, 5 , 8, 4, 1]
j=0	temp<arr[0](2<3)	true	space shift	[○ , 3, 5, 8, 4, 1]
j=-1	while loop failed (j<=0)		data inserted	[2, 3, 5, 8, 4, 1]

Pass 4: **our array is arr = [2, 3, 5, 8, 4, 1]**

i = 4 temp=arr[4] =4 j = 4 - 1= 3

j=3	temp<arr[3](4<8)	true	data picked	[2, 3, 5, ○ , 8, 1]
j=2	temp<arr[2](4<5)	true	space shift	[2 ,3 ,○ ,5, 8, 1]
j=1	temp<arr[1](4<3)	false	data inserted	[2 ,3 ,4 ,5, 8, 1]

Pass 4: **our array is arr = [2 ,3 ,4 ,5, 8, 1]**

i = 5 temp=arr[5] =1 j = 5 - 1= 4

j=4	temp<arr[4](1<8)	true	data picked and shift	[2 ,3 ,4 ,5, ○, 8]
j=3	temp<arr[3](1<5)	true	space shift	[2 ,3 ,4 ,○, 5, 8]
j=2	temp<arr[2](1<4)	true	space shift	[2 ,3 ,○ ,4, 5, 8]
j=1	temp<arr[1](1<3)	true	space shift	[2 ,○ ,3 ,4, 5, 8]
j=0	temp<arr[0](1<2)	true	space shift	[○ ,2 ,3 ,4, 5, 8]
j=-1	while loop failed (j<=0)		data inserted	[1 ,2 ,3 ,4, 5, 8]

for loop ended and data is sorted…………………..

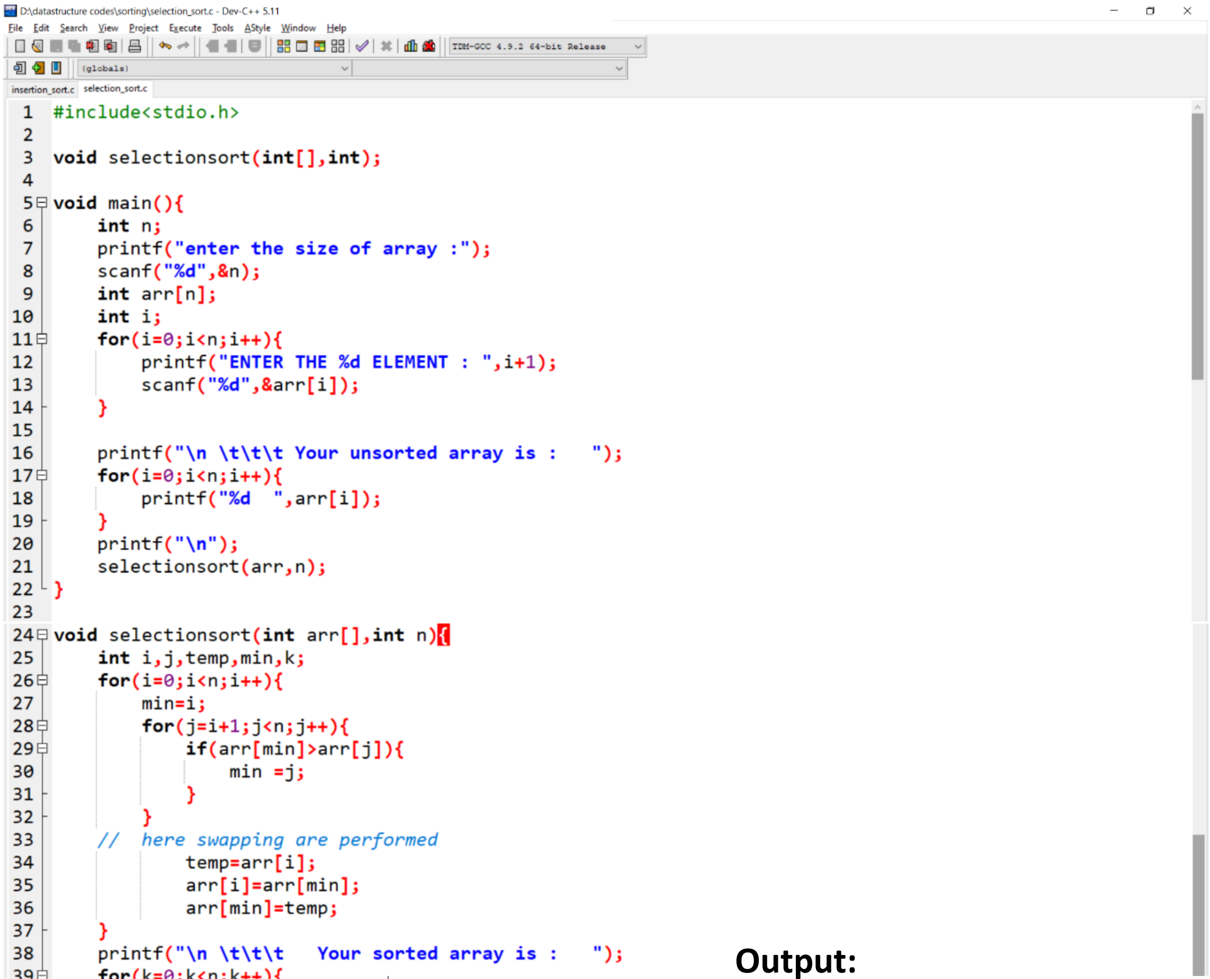
Selection Sort:

- In selection sort again we create two set i.e.: sorted and unsorted set of the array
- First we traverse the array and find the minimum element (for ascending order).
- For traversing we assume that the first element of the unsorted set is small and using linear search we find that whether there is any small element is present or not.
- If there is small element is found then that element is swapped with the small element
- After swapping the sorted set is updated having that element and the unsorted list start with the next element, this process continues until the last element get sorted.
- Why we call it section sort? because it selects a element from the unsorted set compare it to the other element for finding the smallest one and then it swapped it.

Time complexity:

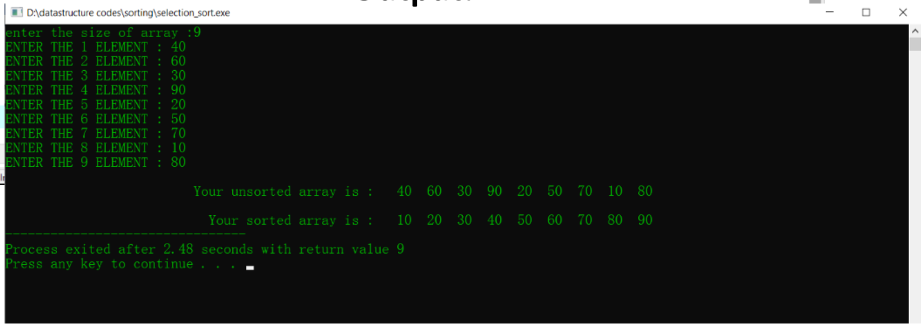
For Best case, worst case, average case $O(n^2)$

Program to implement the selection sort


D:\datastructure codes\sorting\selection_sort.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
insertion_sort.c
selection_sort.c
Compiler
Resources
Compile Log
Debug
Find Results
Line: 44
Col: 2
Sel: 0
Lines: 44
Length: 778

```c
#include<stdio.h>

void selectionsort(int[],int);

void main(){
	int n;
	printf("enter the size of array :");
	scanf("%d",&n);
	int arr[n];
	int i;
	for(i=0;i<n;i++){
		printf("ENTER THE %d ELEMENT  : ",i+1);
		scanf("%d",&arr[i]);
	}
	
	printf("\n \t\t\t Your unsorted array is :   ");
	for(i=0;i<n;i++){
		printf("%d  ",arr[i]);
	}
	printf("\n");
	selectionsort(arr,n);
}

void selectionsort(int arr[],int n){
	int i,j,temp,min,k;
	for(i=0;i<n;i++){
		min=i;
		for(j=i+1;j<n;j++){
			if(arr[min]>arr[j]){
				min =j;
			}
		}
	//  here swapping are performed
			temp=arr[i];
			arr[i]=arr[min];
			arr[min]=temp;
	}
	printf("\n \t\t\t   Your sorted array is :   ");
	for(k=0;k<n;k++){
		printf("%d  ",arr[k]);
	}
	
	printf("\n\n");
}
```

Output:



```
enter the size of array :9
ENTER THE 1 ELEMENT : 40
ENTER THE 2 ELEMENT : 60
ENTER THE 3 ELEMENT : 30
ENTER THE 4 ELEMENT : 90
ENTER THE 5 ELEMENT : 20
ENTER THE 6 ELEMENT : 50
ENTER THE 7 ELEMENT : 70
ENTER THE 8 ELEMENT : 10
ENTER THE 9 ELEMENT : 80

                         Your unsorted array is :   40  60  30  90  20  50  70  10  80

                           Your sorted array is :   10  20  30  40  50  60  70  80  90
--------------------------------
Process exited after 2.48 seconds with return value 9
Press any key to continue . . .
```

For better understanding we take an example and see the flow of execution:

Let us consider an (unsorted) array arr = [3, 8, 5, 2, 4, 1] here n=6(size of array)

🟦 → unsorted set
🟨 → sorted set

Pass 1: **our array is arr = [3, 8, 5, 2, 4, 1]**

i = 0 min = i and j = i + 1 hence j starts with '1'

min = 0	j =1	arr[min] > arr[j] (3>8)	false	no change in min
min = 0	j =2	arr[min] > arr[j] (3>5)	false	no change in min
min = 0	j =3	arr[min] > arr[j] (3>2)	true	change min = 3
min = 3	j =4	arr[min] > arr[j] (2>4)	false	no change in min
min = 3	j =5	arr[min] > arr[j] (2>1)	true	change min = 5

loop in j ends and we get the min value at place 5 of this array

swap arr[i] with arr[min] such that arr[0] with arr[5] now **arr = [1 , 8, 5, 2, 4, 3]**

Pass 2: **our array is arr = [1 , 8, 5, 2, 4, 3]**

i = 1 min = i and j = i + 1 hence j starts with '2'

min = 1	j =2	arr[min] > arr[j] (8>5)	true	change min = 2
min = 2	j =3	arr[min] > arr[j] (5>2)	true	change min = 3
min = 3	j =4	arr[min] > arr[j] (2>4)	false	no change in min
min = 3	j =5	arr[min] > arr[j] (2>1)	false	no change in min

loop in j ends and we get the min value at place 3 of this array

swap arr[i] with arr[min] such that arr[1] with arr[3] now **arr = [1 , 2, 5, 8, 4, 3]**

Pass 3: **our array is arr = [1 , 2, 5, 8, 4, 3]**

i = 2 min = i and j = i + 1 hence j starts with '3'

min = 2	j =3	arr[min] > arr[j] (5>8)	false	no change in min
min = 2	j =4	arr[min] > arr[j] (5>4)	true	change min = 4
min = 4	j =5	arr[min] > arr[j] (4>3)	true	change min = 5

loop in j ends and we get the min value at place 5 of this array

swap arr[i] with arr[min] such that arr[2] with arr[5] now **arr = [1 , 2, 3, 8, 4, 5]**

Pass 4: **our array is arr = [1 , 2, 3, 8, 4, 5]**

i = 3 min = i and j = i + 1 hence j starts with '4'

min = 3	j =4	arr[min] > arr[j] (8>4)	true	change min = 4
min = 4	j =5	arr[min] > arr[j] (4>5)	false	no change in min

loop in j ends and we get the min value at place 4 of this array

swap arr[i] with arr[min] such that arr[3] with arr[4] now **arr = [1 , 2, 3, 4, 8, 5]**

Pass 4: **our array is arr = [1 , 2, 3, 4, 8, 5]**

i = 4 min = i and j = i + 1 hence j starts with '5'

min = 4	j =5	arr[min] > arr[j] (8>5)	true	change min = 5

loop in j ends and we get the min value at place 5 of this array

swap arr[i] with arr[min] such that arr[4] with arr[5] now **arr = [1 , 2, 3, 4, 5, 8]**

finally the i loop exits with having the array **arr = [1 , 2, 3, 4, 5, 8]** which is sorted.

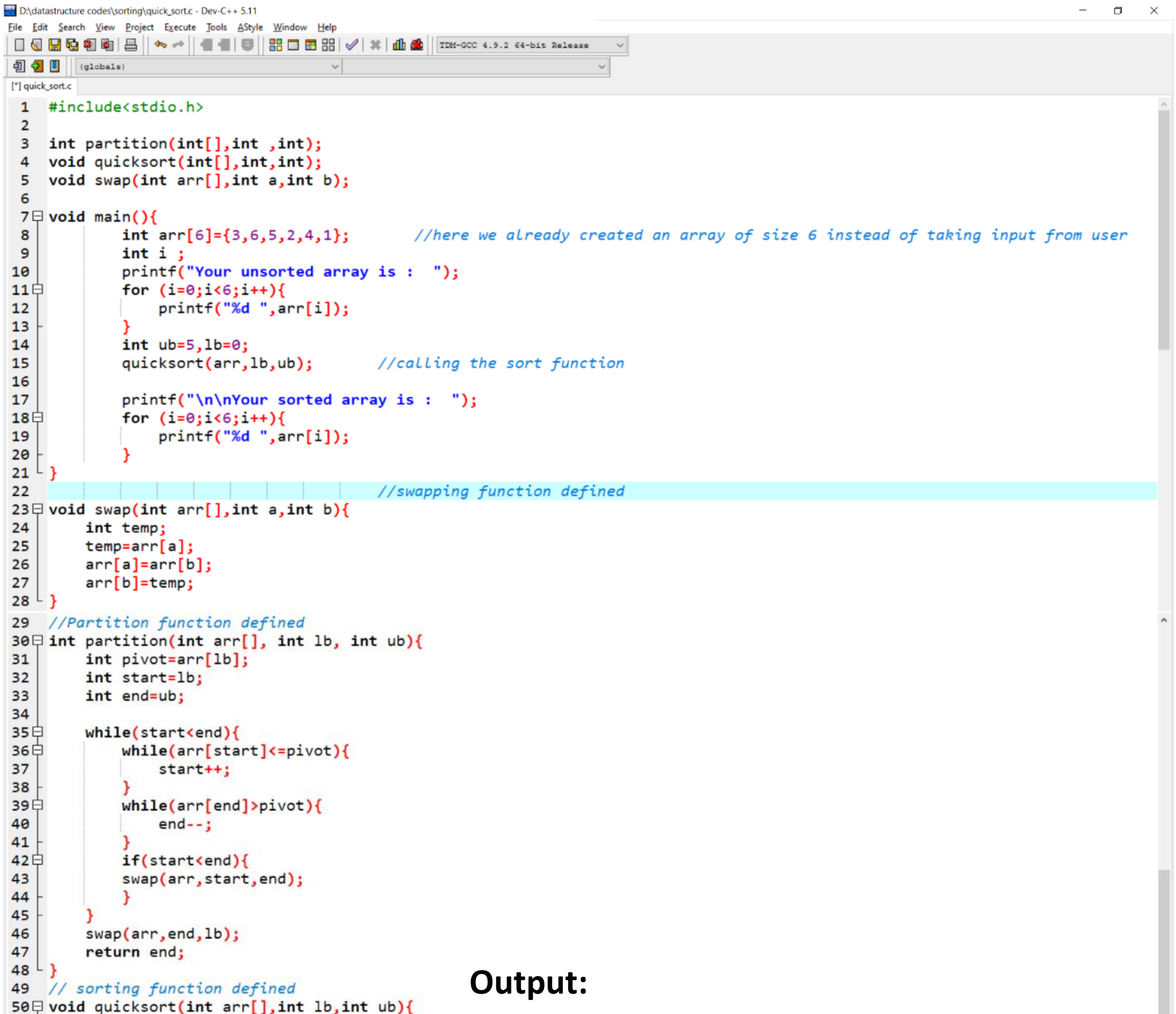
DATA STRUCTURE NOTES

Quick Sort:

- It is based on divide and conquer mechanism.
- First, we have to set the upper bound i.e. 0 and lower bound i.e. size_of_array - 1 and choose a pivot point any of your choice (first, mid or last element). here I choose the first element as my pivot point
- After that we have create a **partition function which is backbone** of this sorting technique. In partition function we can arrange our data in such a manner that the left side of pivot is smaller data and right side of pivot is greater, this function return that value of that index from where we have to divide our array.
- The index returned from the partition function is feed into the quicksort with some logic for recursive call of quicksort
- The quicksort function recursively call itself until each element became single. The *base case* of the recursion occurs when the array has zero or one element because in that case the array is already sorted.

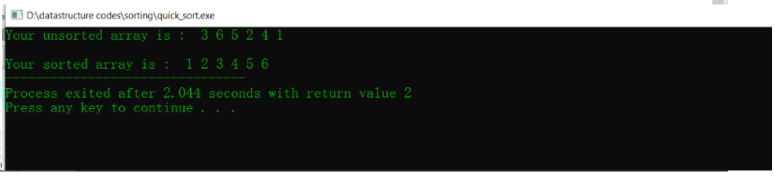
Time complexity:	
Best case ,Average case : *O* (n logn)	Worst case :*O* (n^2)

Program to implement the quick sort


D:\datastructure codes\sorting\quick_sort.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
[*] quick_sort.c
Compiler Resources Compile Log Debug Find Results
Line: 55 Col: 31 Sel: 0 Lines: 57 Length: 1165 Insert Done pa

```
#include<stdio.h>

int partition(int[],int ,int);
void quicksort(int[],int,int);
void swap(int arr[],int a,int b);

void main(){
        int arr[6]={3,6,5,2,4,1};       //here we already created an array of size 6 instead of taking input from user
        int i ;
        printf("Your unsorted array is :  ");
        for (i=0;i<6;i++){
            printf("%d ",arr[i]);
        }
        int ub=5,lb=0;
        quicksort(arr,lb,ub);       //calling the sort function

        printf("\n\nYour sorted array is :  ");
        for (i=0;i<6;i++){
            printf("%d ",arr[i]);
        }
}
                                    //swapping function defined
void swap(int arr[],int a,int b){
    int temp;
    temp=arr[a];
    arr[a]=arr[b];
    arr[b]=temp;
}
//Partition function defined
int partition(int arr[], int lb, int ub){
    int pivot=arr[lb];
    int start=lb;
    int end=ub;

    while(start<end){
        while(arr[start]<=pivot){
            start++;
        }
        while(arr[end]>pivot){
            end--;
        }
        if(start<end){
        swap(arr,start,end);
        }
    }
    swap(arr,end,lb);
    return end;
}
// sorting function defined
void quicksort(int arr[],int lb,int ub){
    int d;
    if (lb<ub){
        d=partition(arr,lb,ub);
        quicksort(arr,lb,d-1);
        quicksort(arr,d+1,ub);
    }
}
```

Output:



```
Your unsorted array is :  3 6 5 2 4 1

Your sorted array is :  1 2 3 4 5 6

--------------------------------
Process exited after 2.044 seconds with return value 2
Press any key to continue . . .
```

Written by:
SAMUNDAR SINGH

For better understanding we take an example and see the flow of execution:

Let us consider an (unsorted) array arr = [3, 8, 5, 2, 4, 1] here n=6(size of array)

Here we consider:

start = lb = 0 end = ub(size–1) pivot = a[lb](here I am assuming the first element as pivot)

Pass 1: our array is a = [3, 8, 5, 2, 4, 1] start = 0 end = 5 pivot = 3

Main while (start < end) (0<5) true

Inside while (a[start]<=pivot)

Start = 0	a[0]<=pivot (3<=3)	true	start ++
Start = 1	a[1]<=pivot(8<=3)	false	loop exits

Inside while(a[end]>pivot)

end = 5	a[5]>pivot(1>3)	false	loop exits

if(start<end) i.e. (1<5) true (swap data of start with end)

Swap(a[start] with a[end]) i.e. swap a[1] with a[5]

NOW [3, 1, 5, 2, 4, 8]

↑ start ↑ end

Pass 2: our array is a = [3, 1, 5, 2, 4, 8] start = 1 end = 5 pivot = 3

Main while (start < end) (1<5) true

Inside while (a[start]<=pivot)

Start = 1	a[1]<=pivot (1<=3)	true	start++
Start = 2	a[2]<=pivot(5<=3)	false	loop exits

Inside while(a[end]>pivot)

end = 5	a[5]>pivot(8>3)	true	end –
end = 4	a[4]>pivot(4>3)	true	end –
end = 3	a[3]>pivot(2>3)	false	loop exits

if(start<end) i.e. (2<3) true (swap data of start with end)

Swap(a[start] with a[end]) i.e. swap a[2] with a[3]

NOW [3, 1, 2, 5, 4, 8]

↑ start ↑ end

Pass 3: our array is a = [3, 1, 2, 5, 4, 8] start = 2 end = 3 pivot = 3

Main while (start < end) (1<5) true

Inside while (a[start]<=pivot)

Start = 2	a[2]<=pivot (2<=3)	true	start++
Start = 3	a[3]<=pivot(5<=3)	false	loop exits

Inside while(a[end]>pivot)

end = 3	a[3]>pivot(5>3)	true	end –
end = 2	a[2]>pivot(2>3)	false	loop exits

if(start<end) i.e. (3<2) false (No swapping performed)

Pass 4: **our array is a = [3, 1, 2, 5, 4, 8]** **start = 3** **end = 2** **pivot = 3**

Main while (start < end) (3 < 2) False

Now the pivot is swapped with the end and function exits returning the value of the end

Swap(a[lb] with a[end]) i.e. swap(a[0] with a[2])

NOW [2, 1, 3, 5, 4, 8]

Return 2(end value)

a = [2, 1]

a = [5, 4, 8]

for sub array **a=[2,1]** the end point modify end = return value from partition fun -1

Here we consider:

start = lb = 0 end = 2 - 1 = 1 pivot = a[lb](here I am assuming the first element as pivot)

Pass 5: **our array is a = [2, 1]** **start = 0** **end = 1** **pivot = 2**

```
Main while (start < end) (0<1)    true
    Inside while (a[start]<=pivot)
        Start = 0       a[0]<=pivot (2<=2)       true        start ++
        Start = 1       a[1]<=pivot(1<=2)        false       loop exits
    Inside while(a[end]>pivot)
        end = 1         a[1]>pivot(1>2)          false       loop exits

    if(start<end)    i.e. (1<1)  false           (No swapping performed)
```

Pass 6: **our array is a = [2, 1]** **start = 1** **end = 1** **pivot = 2**

Main while (start < end) (1 < 1) False

Now the pivot is swapped with the end and function exits returning the value of the end

Swap(a[lb] with a[end]) i.e. swap(a[0] with a[1])

NOW [1, 2]

Return 1(end value)

a = [1]

a[2]

(since in quick sort function ub == lb hence recursion stops)

Similarly for sub array a=[5, 4, 8] the start point modified **start = return value +1**

Here we consider:
start = lb = 2 + 1 = 3 end = 0 pivot = a[lb] (here I am assuming the first element as pivot)

Pass 7: **our array is a = [5, 4, 8]** **start = 3** **end = 5** **pivot = 5**

Main while (start < end) (3<5) true

Inside while (a[start]<=pivot)

Start = 3	a[3]<=pivot (5<=5)	true	start ++
Start = 4	a[4]<=pivot(4<=5)	true	start ++
Start = 5	a[5]<=pivot(8<=5)	false	loop exits

Inside while(a[end]>pivot)

end = 5	a[5]>pivot(8>5)	true	end --
end = 4	a[4]>pivot(4>5)	false	loop exits

if(start<end) i.e. (5<4) false (No swapping performed)

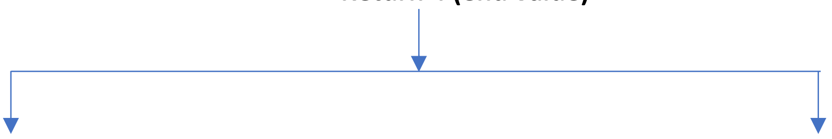
Pass 8: **our array is a = [5, 4, 8]** **start = 5** **end = 4** **pivot = 2**

Main while (start < end) (5 < 4) False

Now the pivot is swapped with the end and function exits returning the value of the end
Swap(a[lb] with a[end]) i.e. swap(a[3] with a[4])

NOW [4, 5 ,8]

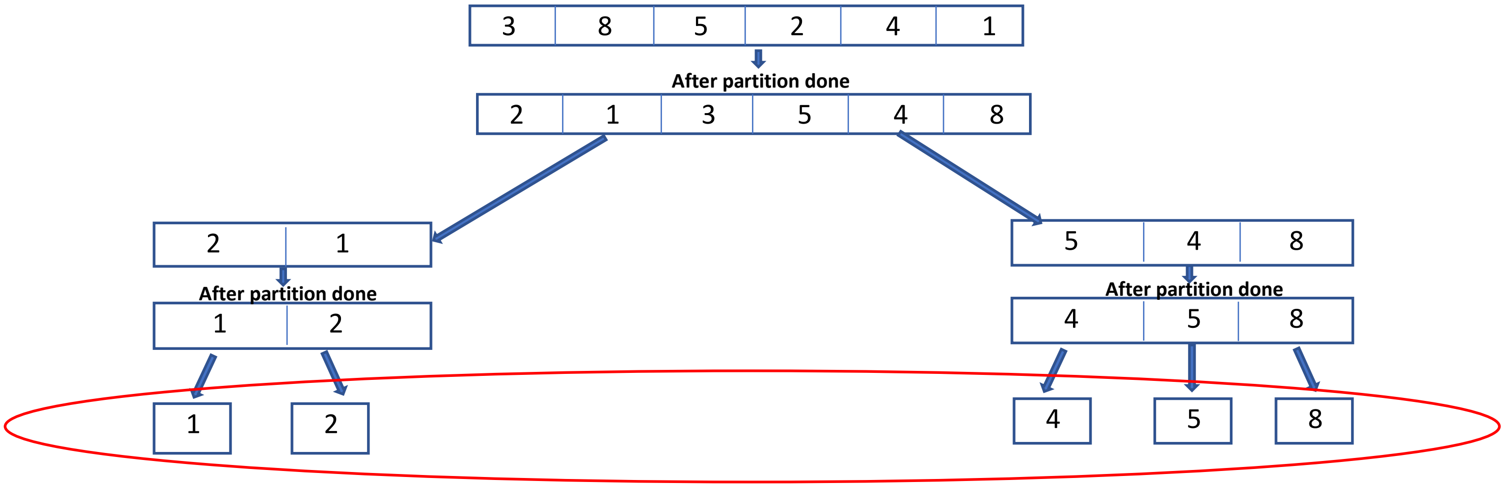
Return 4 (end value)



a[4] a[8]

(since in quick sort function ub == lb hence recursion stops)

Generalising the result from the recursion:



The elements are arranged in ascending order

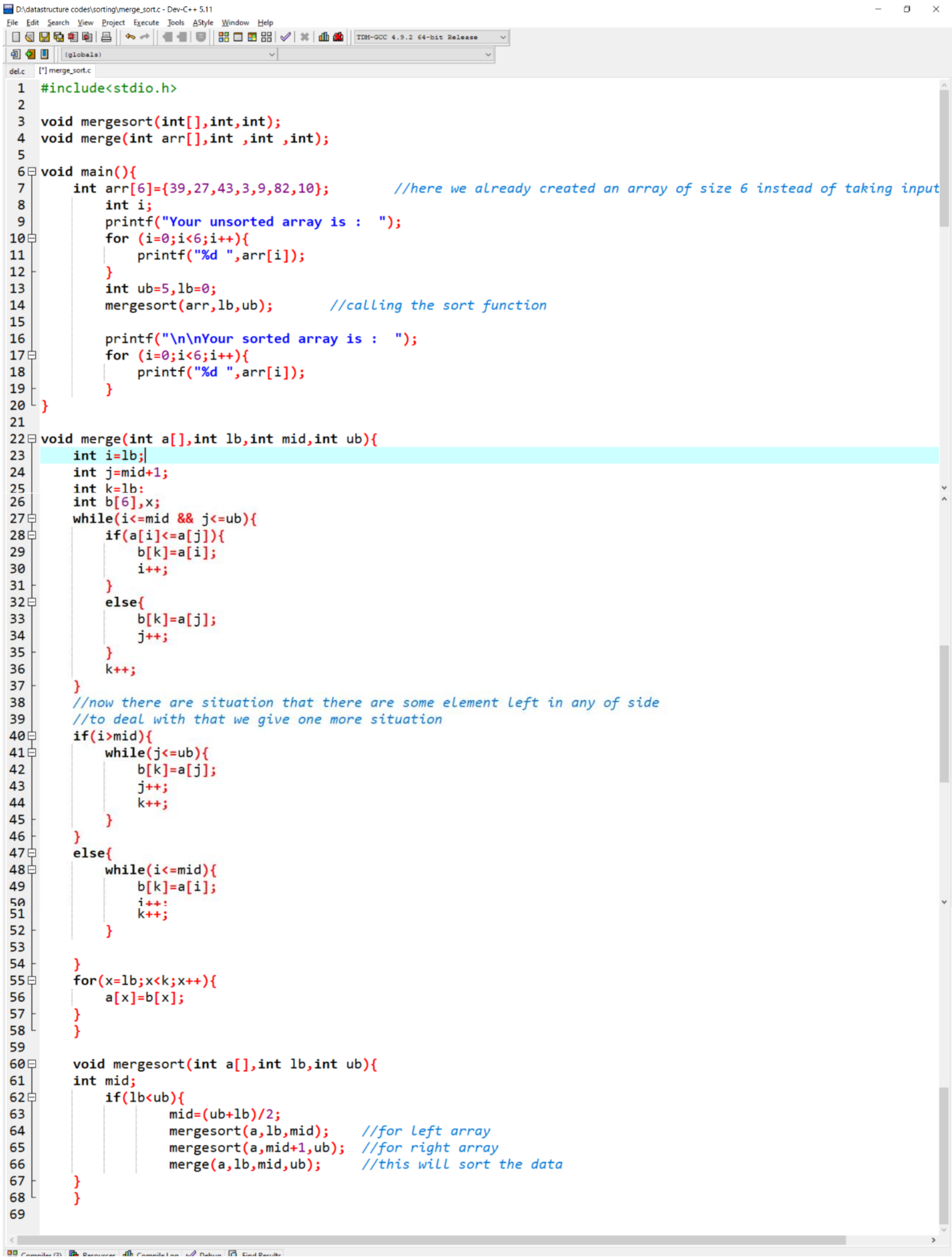
Written by:
SAMUNDAR SINGH

Merge Sort:

- It is base on divide and conquer method to sort the data.
- It take a data from the left sub array compare it with the right sub array the data which is smaller is put in another array
- Here the merge function will be the backbone of this sorting whose work is to compare the sub array and store appropriate data in the temporary array.
- After the sorting is completed then we copy the data of temporary array to the original array.

Time complexity:
Best case ,Average case, Worst case : *O* (n logn)

Program to implement the Merge sort

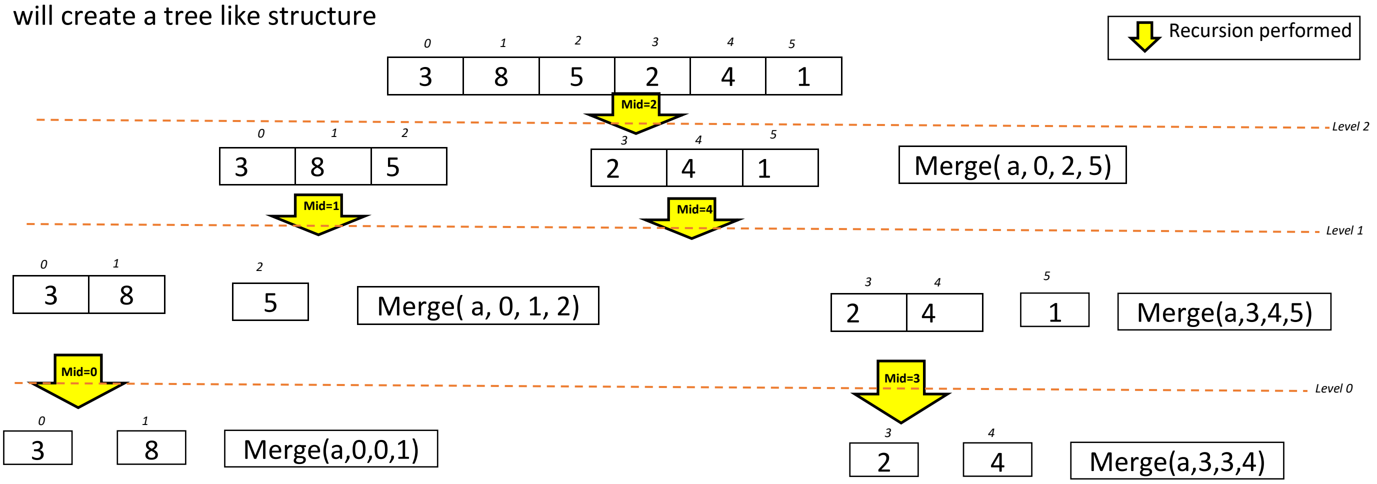

D:\datastructure codes\sorting\merge_sort.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
del.c
[*] merge_sort.c
Compiler (2) Resources Compile Log Debug Find Results

```c
#include<stdio.h>

void mergesort(int[],int,int);
void merge(int arr[],int ,int ,int);

void main(){
    int arr[6]={39,27,43,3,9,82,10};        //here we already created an array of size 6 instead of taking input
        int i;
        printf("Your unsorted array is :  ");
        for (i=0;i<6;i++){
            printf("%d ",arr[i]);
        }
        int ub=5,lb=0;
        mergesort(arr,lb,ub);       //calling the sort function

        printf("\n\nYour sorted array is :  ");
        for (i=0;i<6;i++){
            printf("%d ",arr[i]);
        }
}

void merge(int a[],int lb,int mid,int ub){
    int i=lb;
    int j=mid+1;
    int k=lb;
    int b[6],x;
    while(i<=mid && j<=ub){
        if(a[i]<=a[j]){
            b[k]=a[i];
            i++;
        }
        else{
            b[k]=a[j];
            j++;
        }
        k++;
    }
    //now there are situation that there are some element left in any of side
    //to deal with that we give one more situation
    if(i>mid){
        while(j<=ub){
            b[k]=a[j];
            j++;
            k++;
        }
    }
    else{
        while(i<=mid){
            b[k]=a[i];
            i++;
            k++;
        }

    }
    for(x=lb;x<k;x++){
        a[x]=b[x];
    }
    }

    void mergesort(int a[],int lb,int ub){
    int mid;
        if(lb<ub){
                mid=(ub+lb)/2;
                mergesort(a,lb,mid);    //for left array
                mergesort(a,mid+1,ub);  //for right array
                merge(a,lb,mid,ub);     //this will sort the data
    }
    }

```

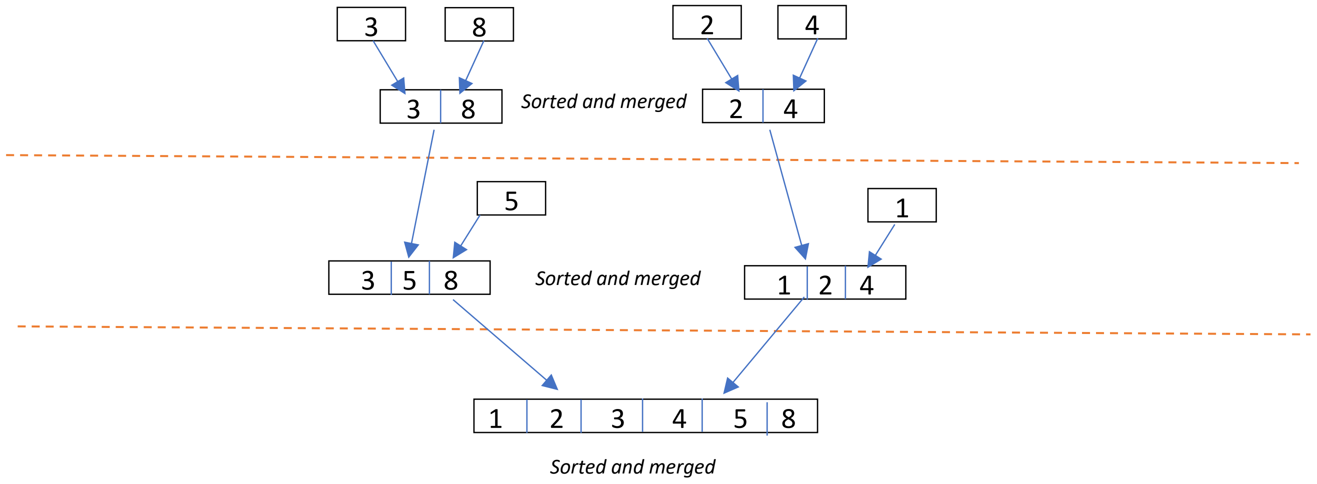
For better understanding we take an example and see the flow of execution:

Let us consider an (unsorted) array arr = [3, 8, 5, 2, 4, 1] here n=6(size of array)

At first the recursion will carry on and the whole array is divided into sub array until the base class stop it so clearly our merge function start only when the whole array is divided into single sub array it will create a tree like structure



After this division the merge function will create a sorted sub array which is going to merge in an array. This merge function start operating from downward to upward direction.

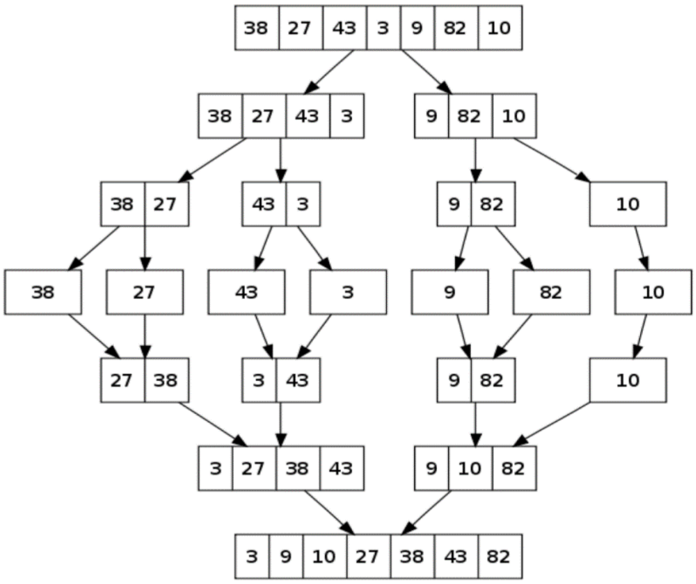


The tressing is quite complex so better to have the explanation of the working of this sorting technique

- ○ The basic steps of a merge sort algorithm are as follows:
- ○ If the array is of length 0 or 1, then it is already sorted.
- ○ Otherwise, divide the unsorted array into two sub-arrays of about half the size.
- ○ Use merge sort algorithm recursively to sort each sub-array.
- ○ Merge the two sub-arrays to form a single sorted list

Written by:
SAMUNDAR SINGH

if we plot an execution map of previous example then



To be continue……………………………………………

DATA STRUCTURE NOTES

Written by:

SAMUNDAR SINGH