School of Computing and Informatics

Data Structure and algorithms

Semester 3, year 2022

Student name: Hamza.Muhsen
Student id: 20110044
Section number: 1
Date: 3-9-2022

TASK ONE:

A)

In this section, I will select linked list over list(array) because linked list because it's better in time and space complexity than the list, the linked list it's the same as array has elements but in linked list, every element called a node and every node divided into two sections -> one called value (value of element) and the other is next (the value of the next element to link the elements to each other). As shown in the figure, this called a linked list. Also as shown in the figure, there is something called (Head) → and it's mean to the first node of the linked list.

In linked list → we have two main operations:

- Insertion
- Remove

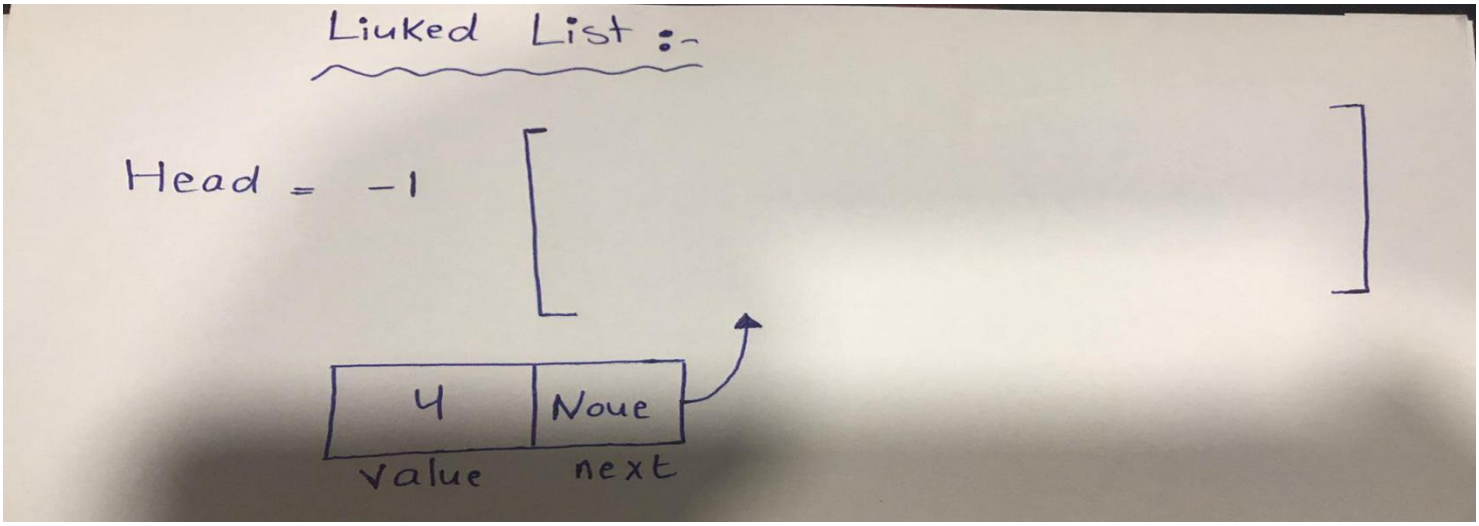
➔ Insertion:

It's the operation that we add the new node to the linked list and link it with the others, in insertion we have three conditions to consider it :

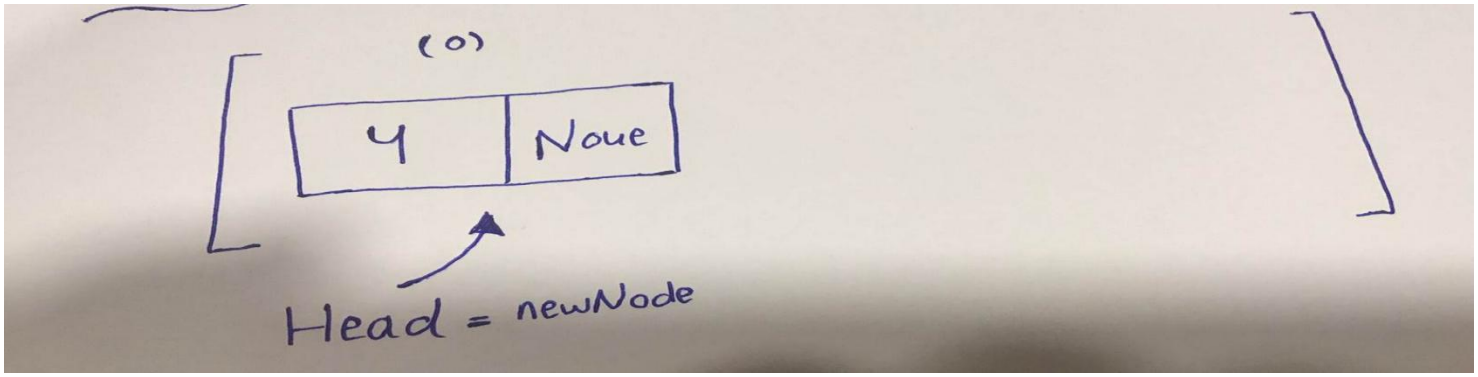
1. Insert in the first of the linked list -> insert(0,value)
2. Insert in the end of the linked list ------> insertend(value)
3. Insert in specific position of the linked list -----> insert (position,value)

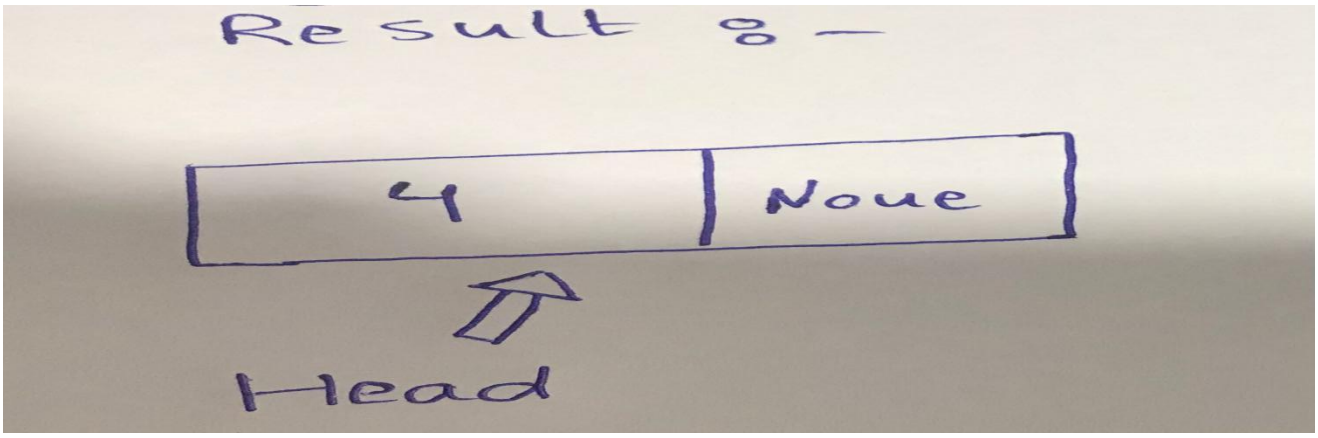
In the first condition, if we want to add a new node in the first of the linked list: insert(0,value)

- Create newnode and add the value that we want to add it in the linked list, I add a (4) value to the new node.If the list is empty (hasn't any elements), the head will be undefined(None), so after we add the value to the newnode, we need to put the newnode inside the array and update the links such as link the head to the newnode.

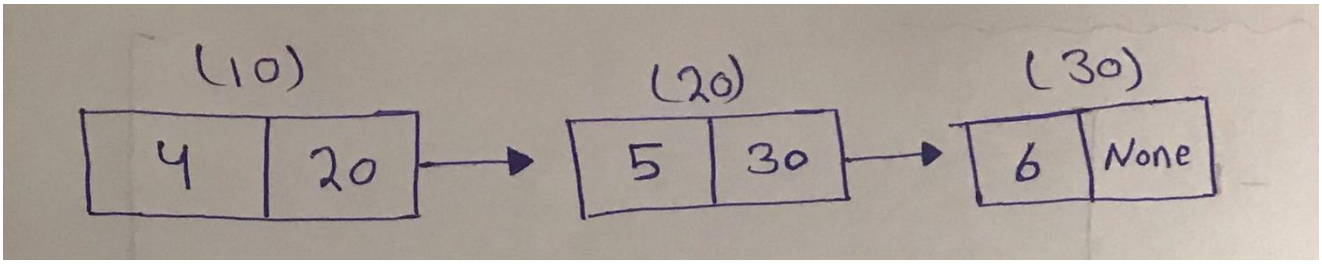


- Make the head indicates to the newnode by changing the value of the head because the first node is always the head of the linked list.
- The linked list will become like this:

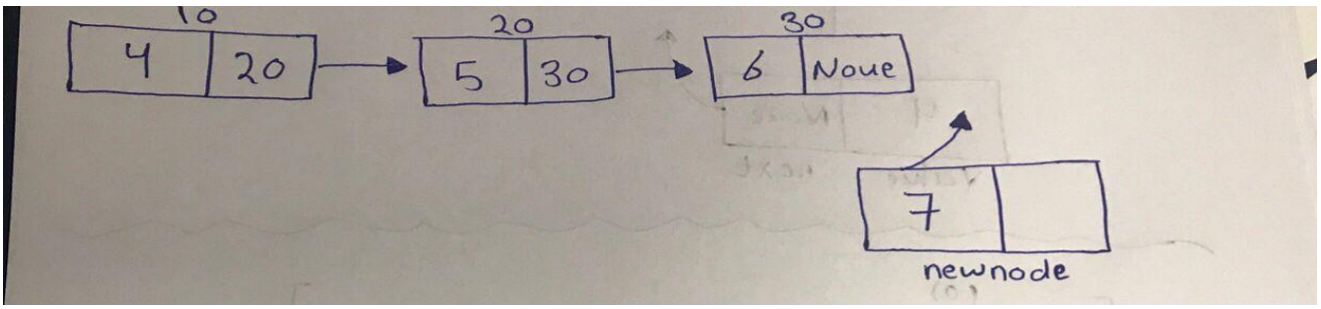




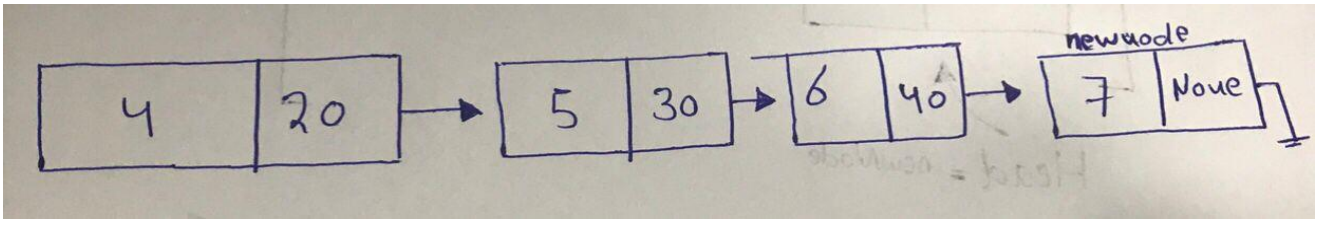
In the second condition, (insert in the end of the linked list): ---> insertend (value)



- Suppose we have this linked list, and we want to insert newnode at the end of the list.



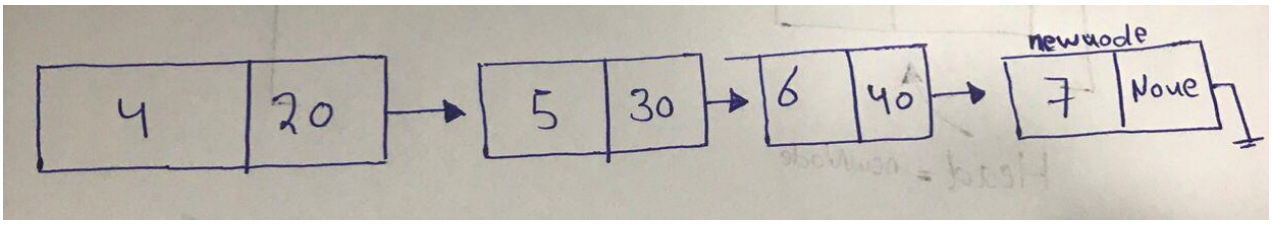
- First, we need to make a newnode with the new value (I put (7) the value of the newnode) and then we need to update the links (reference) so we want to go to the last node in the linked list and change the value of (next) to indicates to the newnode and that's how to update the links in the linked list.



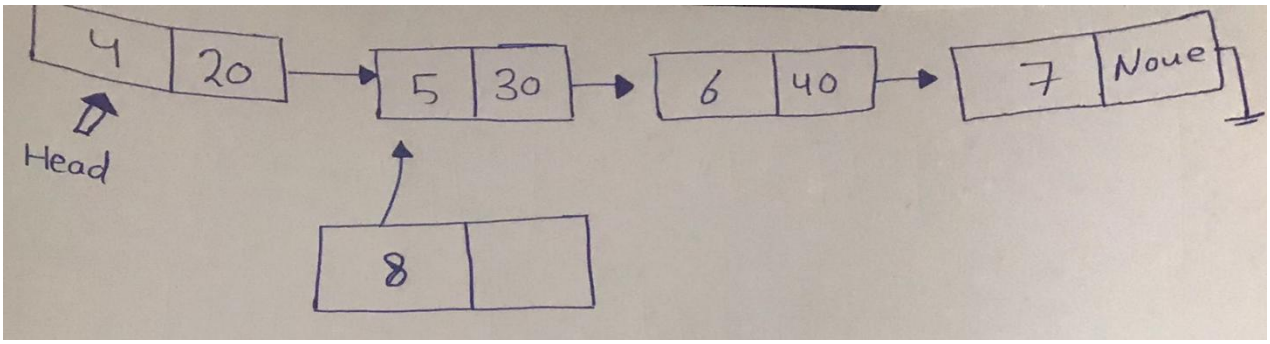
- This is the result of the insert, and we will not forget to mention the head which is at the beginning of the list (first node [4,20]) because the head node will help us in the

implementation stage to write a code to reach the last the node of the linked list and change the next value to the newnode.

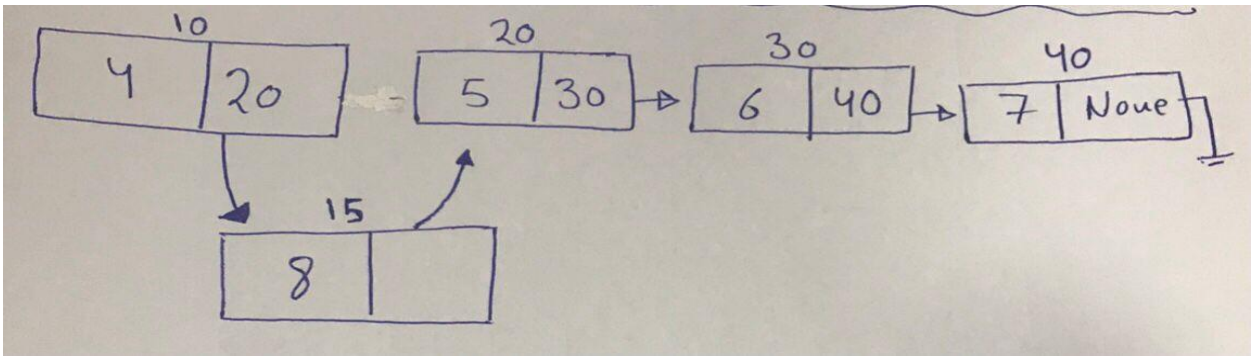
In the last condition (Insert at any valid position in the list): ---> insert (position , value)



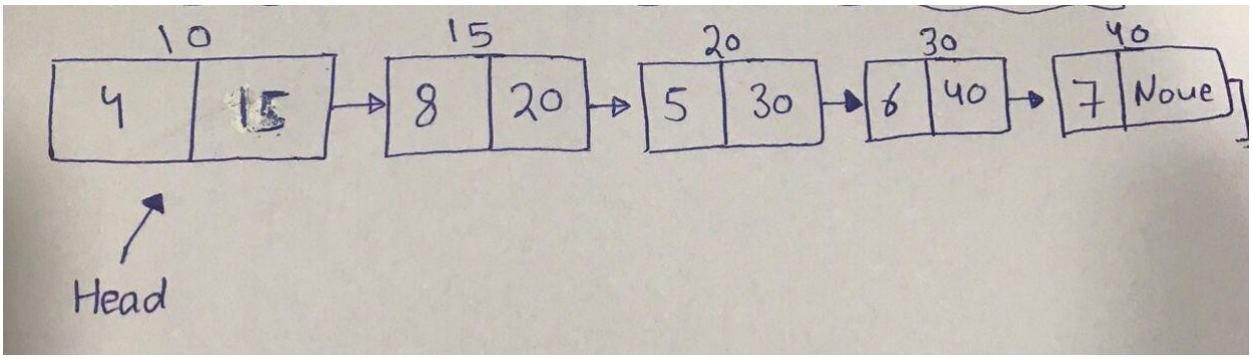
- If we have this list and we need to insert newnode in any position (for example in the index number one[5,30]).



- First, make a newnode and fill it with the value (I put number (8) as a value).



- Change the links (references, change the value of the (next)) of two nodes (previous node and the next node as shown in the figure) by changing the value of the next to the newnode in previous node and changing the next value of the newnode to the next node to make the links.



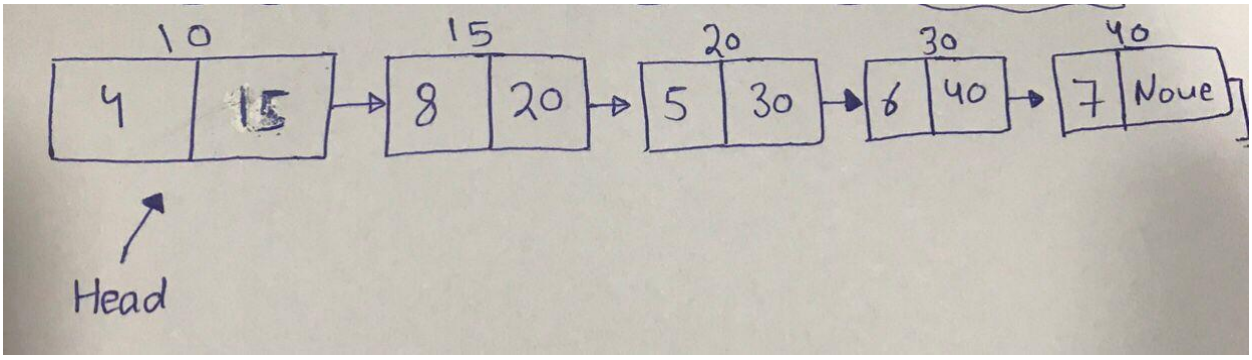
- The result will be like this.

➔ Remove " Remove (position) ":

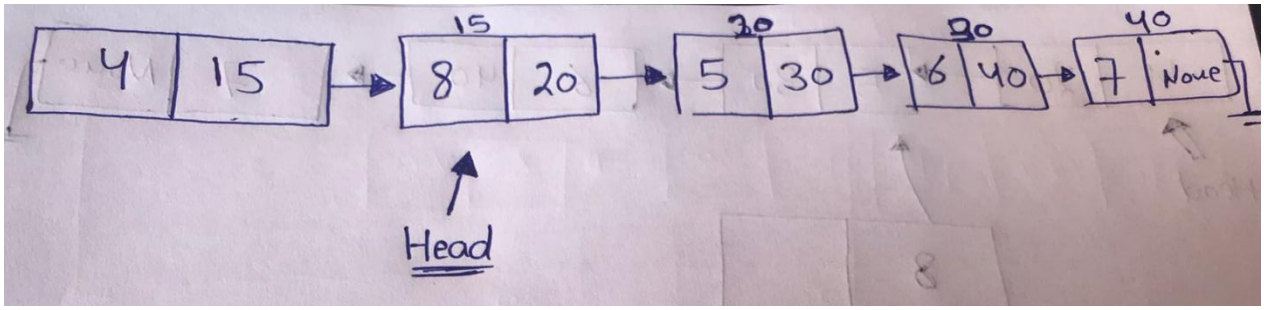
By meaning remove in linked list is to delete the node from the linked list by deleting all the links that indicates to the node. In remove we have two conditions to consider it:

1. Remove the head node. --> Remove(0)
2. Remove at any position in the linked list.--> Remove(position)

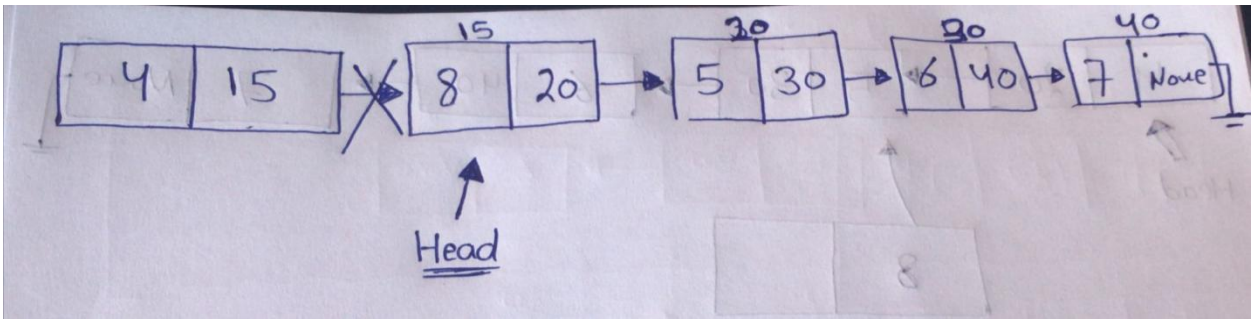
Remove the head node



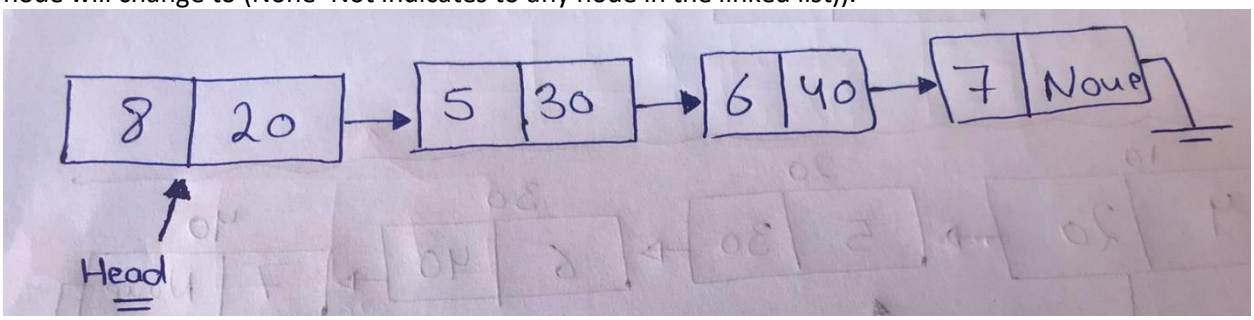
Suppose we have this nodes and we want to remove the head node, there are two important things will change when we remove the head node: the first is the head will change and it will indicates to the second node, the second things is remove the the link between the node we want to remove and the next node.



So first step is changing the head and make it indicates to the next node because the next node (18,20) will be the first node in the linked list after removing the node.

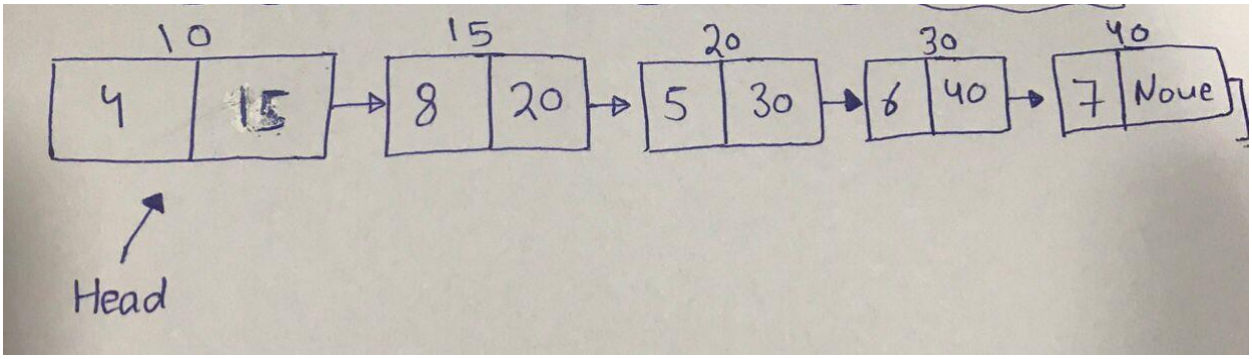


The second step is removing the link between the previous head node and new head node to make the new head node the first node in the linked list.(To remove the link, the (next) value of the previous head node will change to (None=Not indicates to any node in the linked list)).

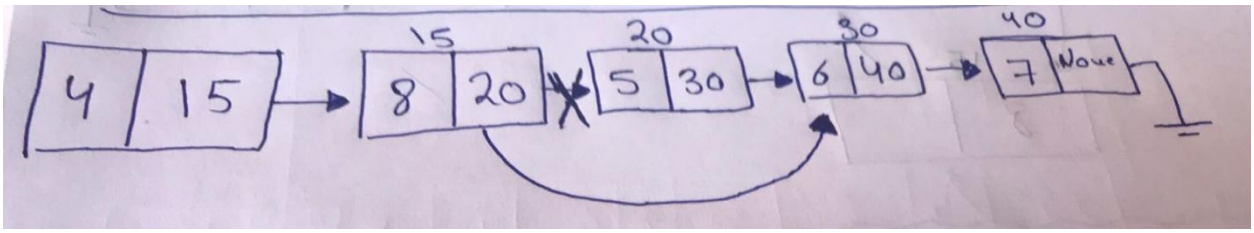


The new linked list.

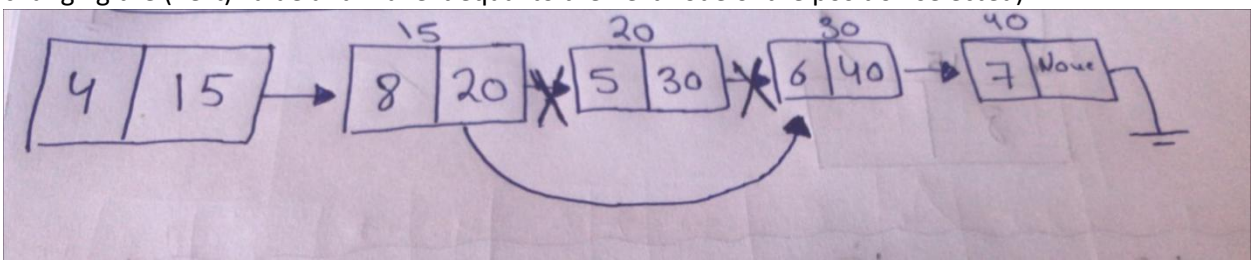
Remove at any position in the linked list



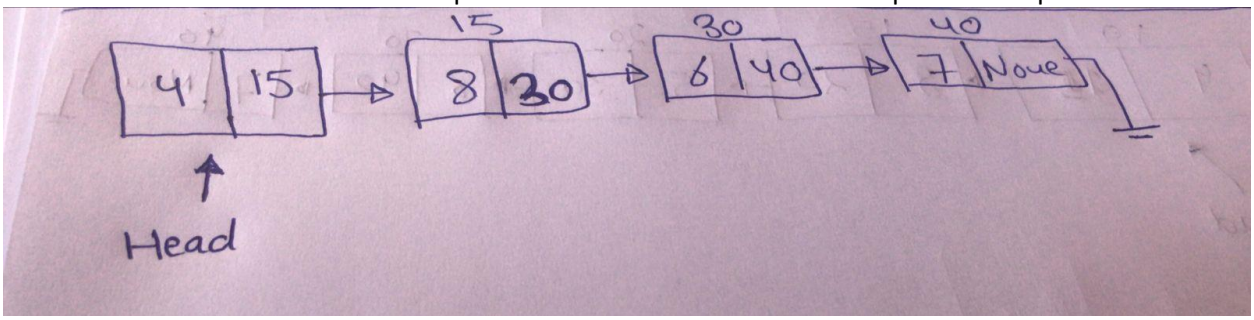
Let's take the last example in this condition, if we want to remove for example node at position (2) in the linked list.



First step, the link between the previous node of the position selected and node selected will remove and the previous node of the position selected will link with the next node of the position selected (by changing the (next) value and make it equal to the next node of the position selected)



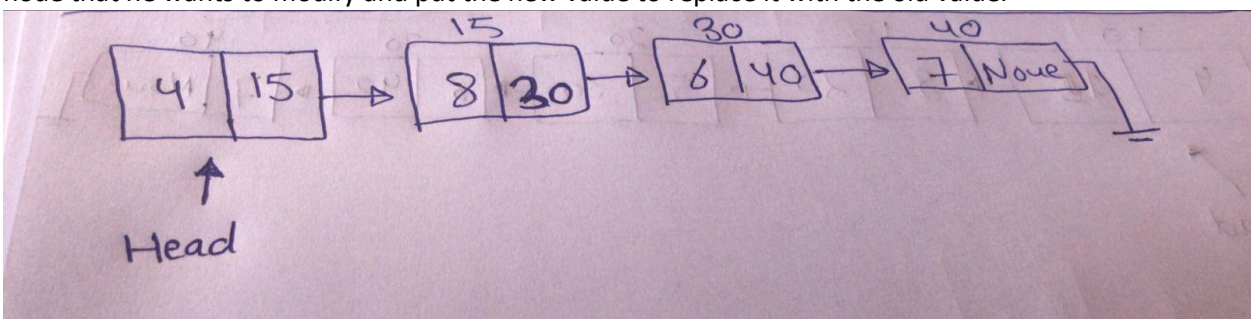
Second step, the link between the node selected and the next node of the position selected will remove (by changing the (next) value of the node selected and make it equal to None) because there is new link that indicates to the next node of the position selected as discussed in the previous step.



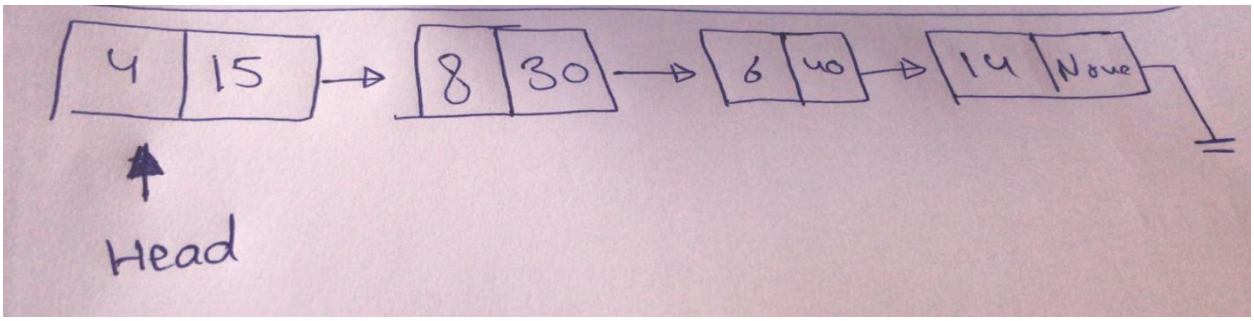
The new linked list.

➔ Operation 3: Modify " Modify (position , new value) "

This operation give the user to modify the value of any node in the linked list by selecting the position of node that he wants to modify and put the new value to replace it with the old value.



Suppose this is a linked list and we want to modify the value of the node at position (3) in the linked list to become the value = 14 instead of 7 .



In the selected node, change the value of the node to 14 .

➔ Operation 4: print " Print () "

This operation is to print the values of each node and show it to the users if the user calls the function (print).

➔ About time complexity in linked list :

Before talk about the time complexity in linked list, we should know about time complexity and what's mean "time complexity", it is the time needed to execute all the code statements in algorithm with respect of the length input. The time complexity is always stated by using the big O notation.

There are many things that we should consider it when we using the linked list (About time comp):

- **Accessing to a element** link list will be O(n) (Worst case).

- Also in **print** operation will take a O(n).

-- **Insertion**:
-In the first condition--> if we want to add a new node in the first of the linked list will take O(1).
-In the second condition--> insert in the end of the linked list (insertend (value)) will take O(n).
-In the last condition --> Insert at any valid position in the list (insert (position , value)) will take O(n).
So in insertion, it will take time O(n) because of worst case.

--**Remove**:
-Remove the head node will take O(1).
-Remove at any position in the linked list will take O(n).
So in remove, it will take also O(n) because of worst case.

-**Modify** also will take a time O(n).

B)

Queue is a concept in data structure describes that first one enter -> first one who take the service (first out), that's why we called it (FIFO). Also in queue, there are two things that we must know :
- Front --> first index in the list, first one in , something like (head) in the linked list.
- Rear--> last index in the list.

In Queue, if we want to add element, the element will add in the rear side and the rear will be the new element. But if we want to delete element, the element will delete in front side and the front will be the next element of the previous front .

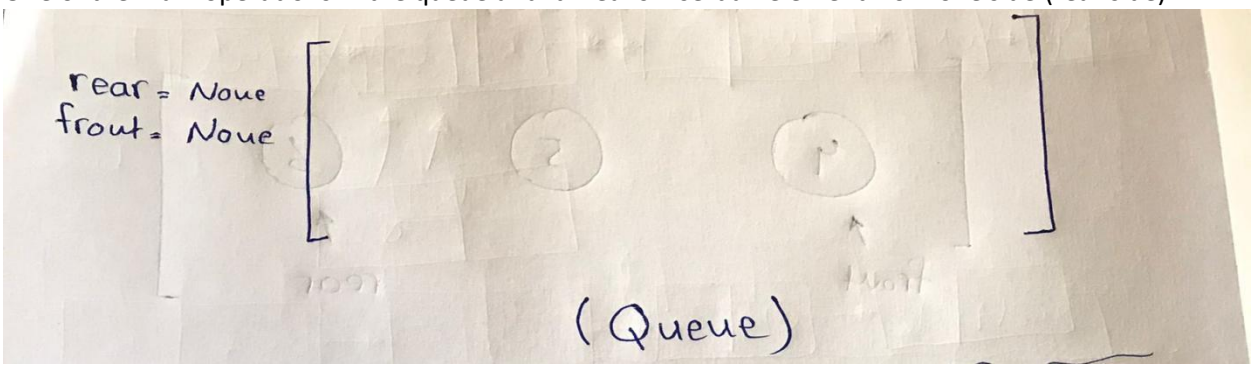
Queue Data Structure

In Queue, we have four operations which are :
- enQueue(item) ---> **main operation**
- deQueue() ---> **main operation**
- isempty()
- front()

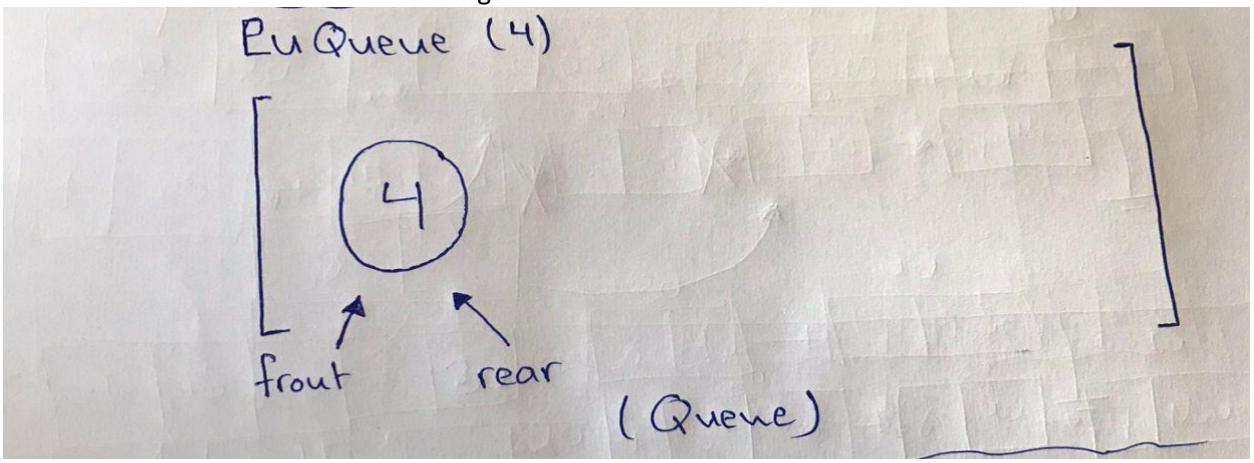
Notice In queue operation , the time complexity must take only O(1).

- enQueue(item)

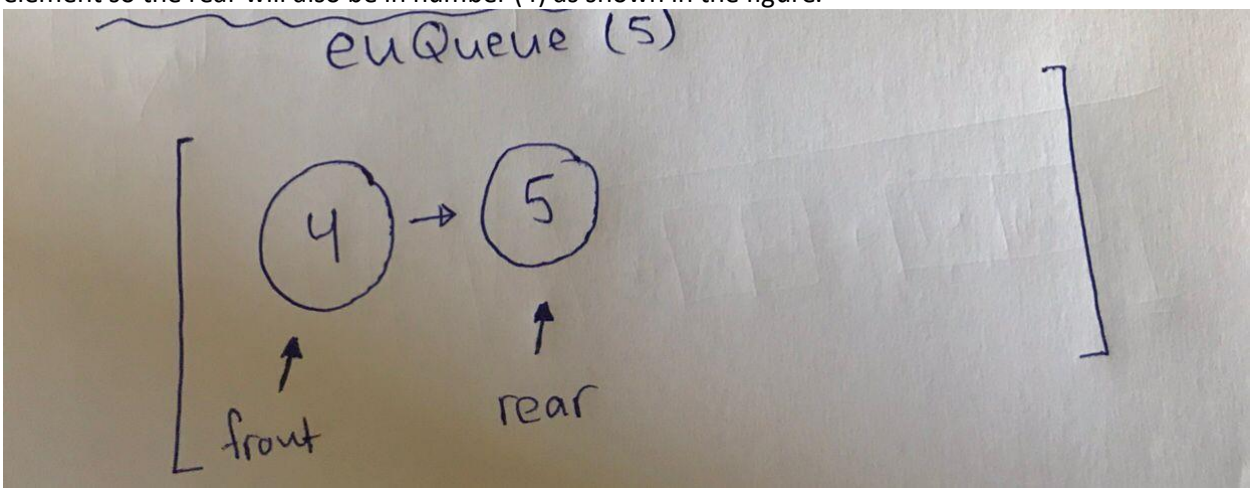
One of the main operations in the queue and it means insert an element from one side (rear side).



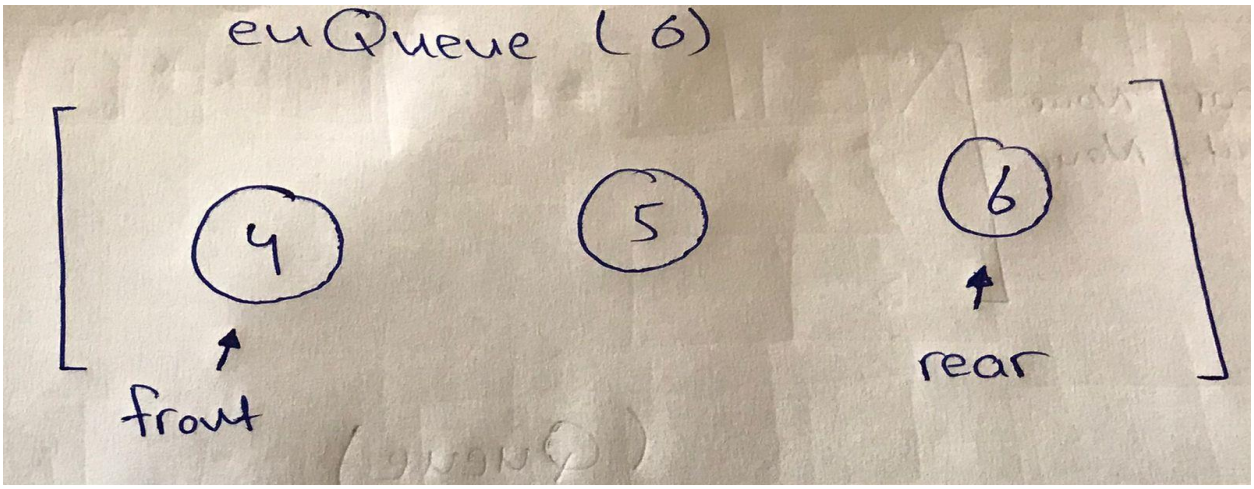
Suppose we have this queue and we want to insert elements in this queue because the queue is empty. Notice that the front and rear are none because the queue is empty but when we insert the element the value of the front and the rear will change.



If we call the function enQueue(4) , the number (4) will be in the queue and before insert the number, the queue was empty so number (4) will be the first element in the queue and the first element means the value of the front and also when we insert new element the rear will be indicates to the new element so the rear will also be in number (4) as shown in the figure.



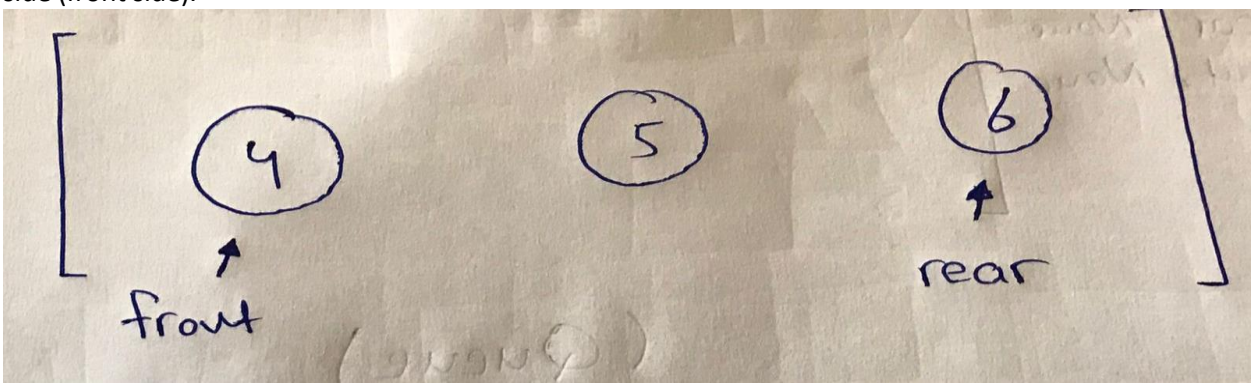
If we call the function again with another number such as (5) , the new element will insert in the queue and the rear will indicate to the new element which is number (5). The front value in insert will not change and it always be the first element in the queue.



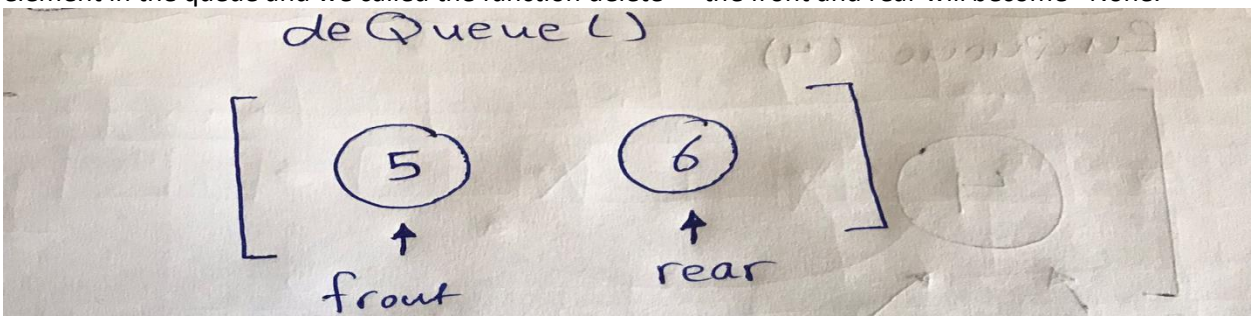
To confirm the idea, if we want to insert another element , the rear will become the new element and the front will not change.

- deQueue() :

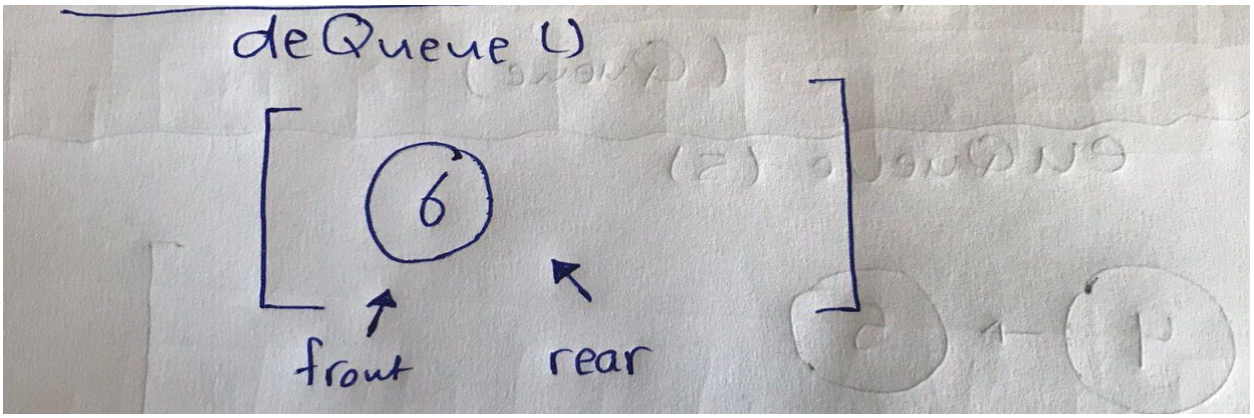
Also it one of the main operations in queue and it means remove element from the other side of insert side (front side).



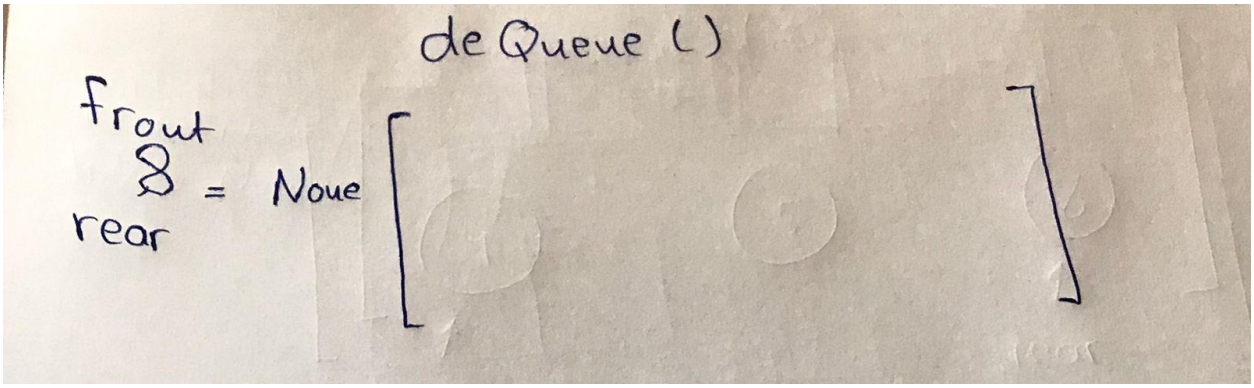
Suppose we have this queue and we want to delete element from the queue, the element will remove from the front side and the front value will become the next element if found, if there is only one element in the queue and we called the function delete--> the front and rear will become= None.



If we call the function deQueue(), the first element (number 4) will delete and the next element (number 5) will be the front of the queue and the rear will not change in delete processes.



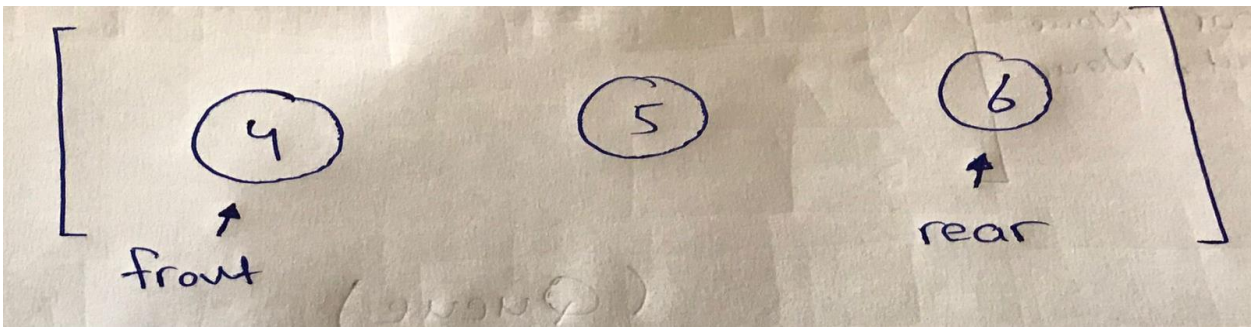
If we call this function again the first element (number 5) will delete and the front will change and it will be the next element (number 6) and also the rear will not change .In this case only the rear and front will be the same value.



If we call the function again, the front and rear will change the value and become (None) because the queue becomes empty and we need to insert element in the queue.

- Front ():

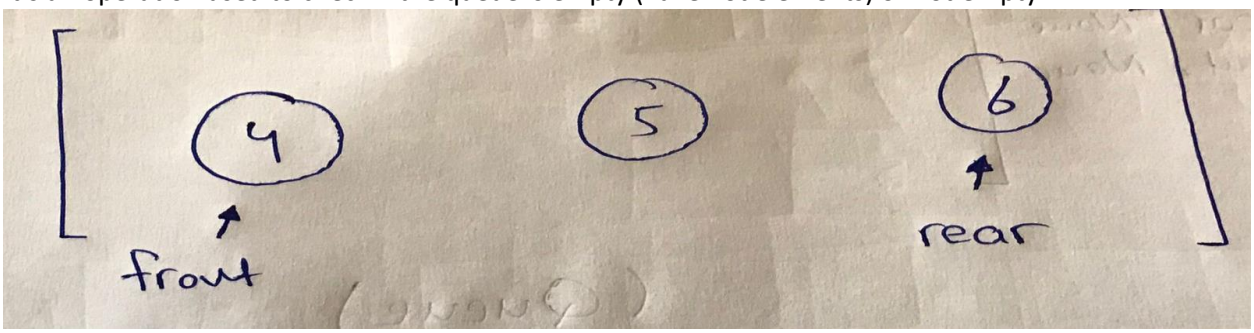
This operations gives the users the value of the front in the queue by returning the value of the front if found.



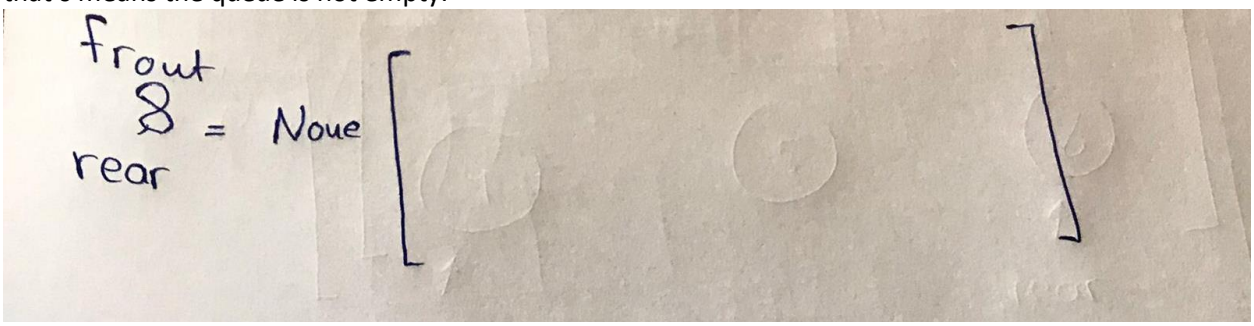
For example if we have this queue and the user called the function front(), the machine will return the (number 4) to the user to know the front value. But if the queue is empty , the error will show to the user because it's empty and the value of front is none.

- Isempty() :

It’s an operation used to check if the queue is empty (have not elements) or not empty.



For example if we have this queue and called the function isempty() the machine will return False and that’s means the queue is not empty.



But if we have this queue and called the function, it will return True and that means the queue id empty.

__Notice__ we can implement this concept (queue) on list or linked list but I prefer the linked list because of space complexity.

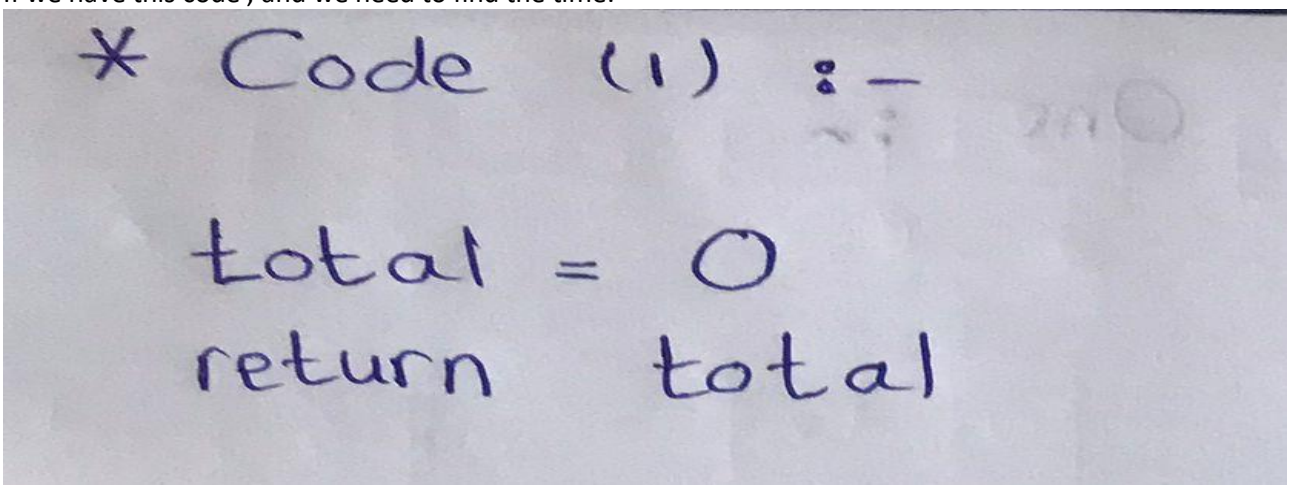
ASYMPTOTIC ANALYSIS

It's a concept in data structure and it means that leads the problems in the analyzing the algorithms. To know more about it and how it benefits in data structure we must know that we evaluate the work of algorithms depending on the input size (in summary: we evaluate on how the time will increase depending on the input size). By using it, we can define the best case, average case and worst case of the algorithms.

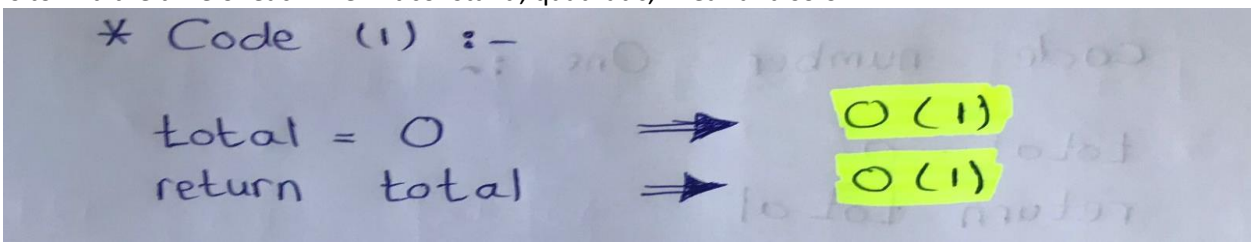
In data strucutre, we will use two tools to describe the time:
- Time complexity -> There are many types of it such as : Linear , Quadratic , Constant and logarithmic time. The best is constant time and the worst is Quadratic.
- Big O notation-> also it is another method to express the time by using O((time) for example : Quadrtic time will be in O notation -> O(n), constant O(1) .

Examples:

If we have this code , and we need to find the time:

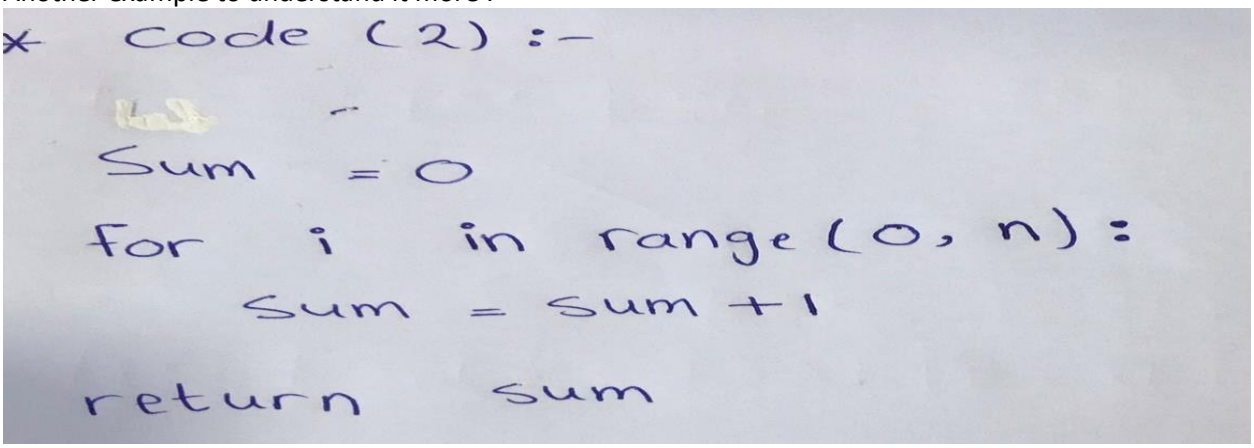


To find the time complexity, we will find the time of each line in the code and add them up. So first step is to find the time of each line if it constant , quadratic, linear and so on .

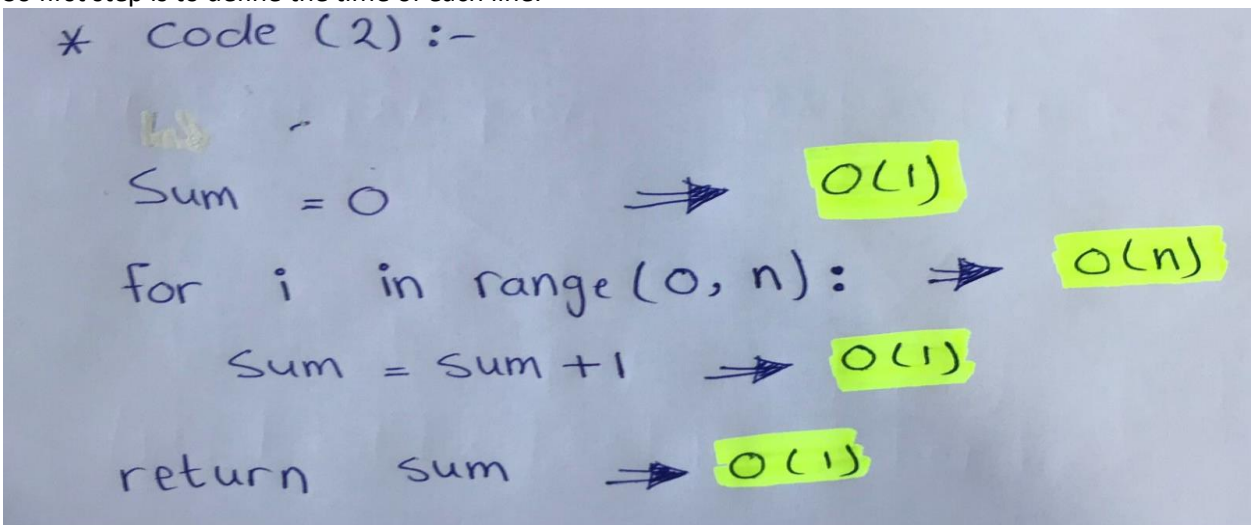


So after we define the time, we will find the total of time:
T=O(1)+O(1) --> notice that there are both constant, constant =c .
T=c1+c2 =c3 so T=O(1) because the total is constant.
So if we get T= O(1)+O(1) -> the total time will always be = O(1).

Another example to understand it more :



So first step is to define the time of each line:



Add them to find the total amount of time:
T=O(1)+ n*O(1)+O(1)+O(1) = c+n*c and always take the worst time which is (n) so the total of time:
T=O(n).

(Asymptotic Analysis (Based on input size) in Complexity Analysis of Algorithms - GeeksforGeeks, 2022)

EFFICIENCY OF AN ALGORITHM

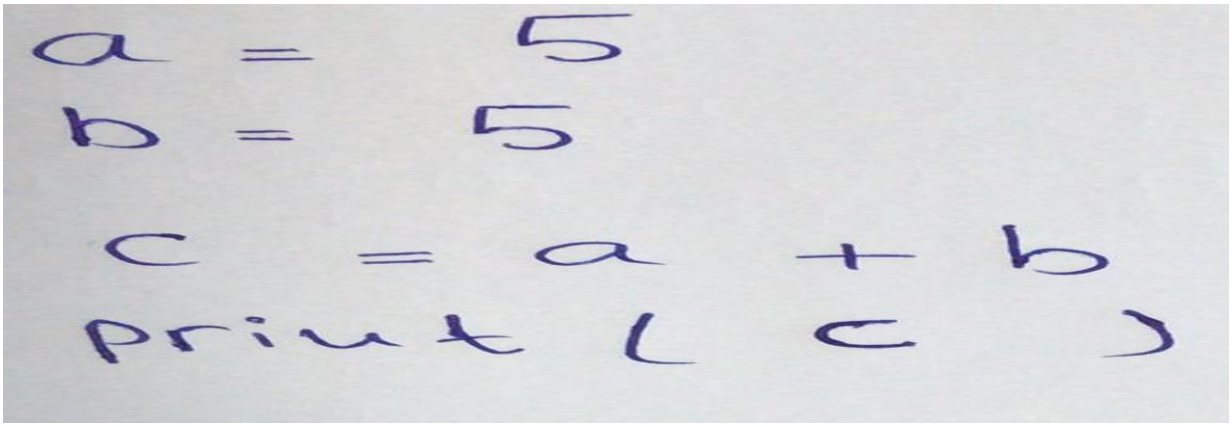
As we learned in data structure, we should not focus only in solving the problem when we writing the code because there is something called efficiency of an algorithm, efficiency of an algorithms are the factors that played an important role in evaluate the algorithm. The main two factors are :

- **Time complexity**

In time, we evaluate the algorithms by different factors. For example --> if we want to define algorithm, we will write a code by using any programming language and execute it. We will evaluate the time taken to execute the code. This time taken is depended on many factors:
- The machine and it's speed.
- Compiler or software used.
- Operating system .
- Programming language.

Also there are many types of the time such as --> Linear time , Constant time , Quadratic time and logarithmic time.We can express it by using big o notation.

Example of time complexity:



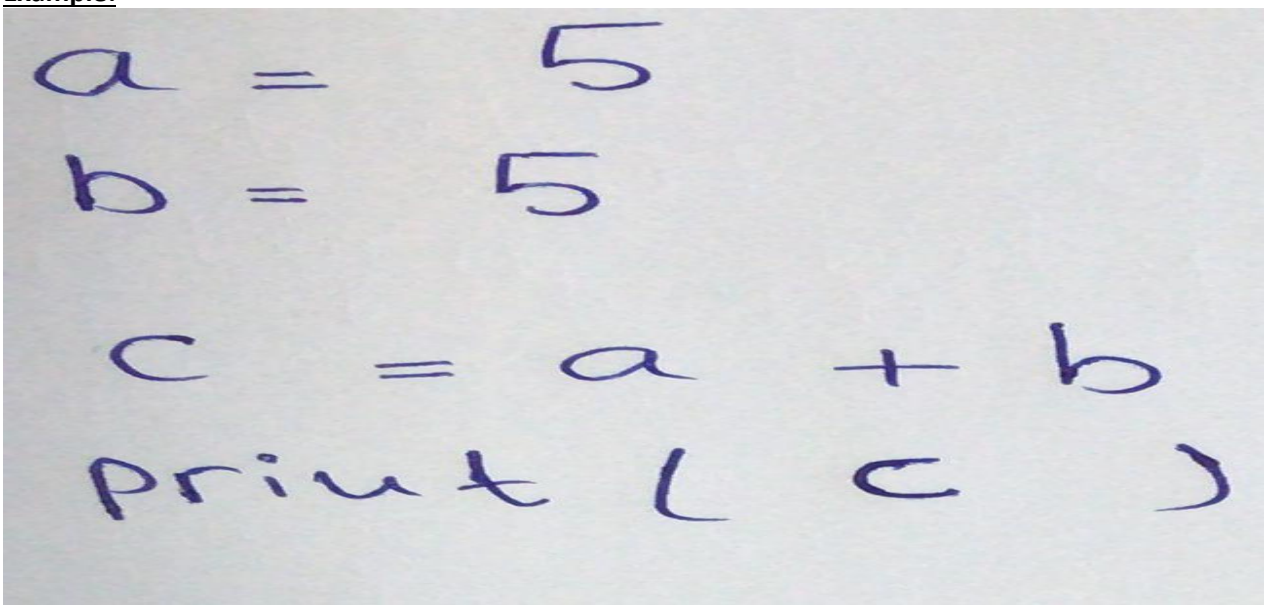
To find the time we will back to the previous question and implement the steps, the time complexity of the code above: O(1).

- **Space complexity**

It is a concept in data structure that means the total of space used by algorithm or code, also in space -> the space of input values are also including in the space. Space complexity is a sensitive factor because if the algorithm takes a lot of space, the compiler will not run the code instead of time if the algorithm takes a lot of time ,the user can wait.

*From the research I do * the general formula to calculate the space complexity is = Auxiliary space + space of input values.

Example:



The same example in the time complexity, if we want to find space complexity:
In the code there are three integers (a,b,c) and every integer takes the size 4 bytes so the space will be =3*4 =12. So this code it also take a space O(1).

(FACE Prep | Land your dream Tech job with FACE Prep, 2022)

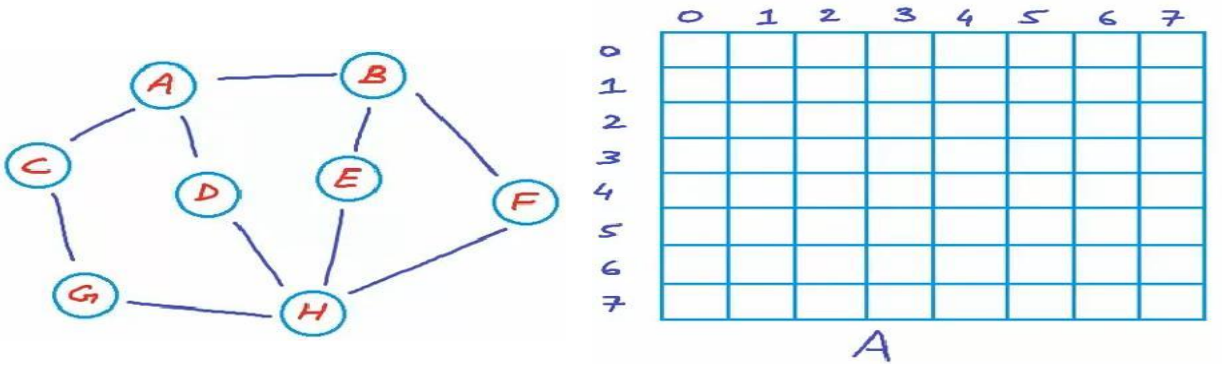
TRADE-OFF WHEN SPECIFYING ADT

Trade-off is a concept in data structure that means --> solve a specific problem by choosing side over another side but the another side will effect the performance when we solve the problem. A big example of trade off is time - space trade off which is we have two options: one) select the less time and big memory(space) two) select the less memory and take a long time. To understand more the time - space let's take an example:

- If the data stored uncompressed--> it will take a big memory but with less time.

Also some examples of the algorithms that use trade off time space is -->

1) dynamic programming, when the time complexity might be less if we use more memory (space).
2) Another example on trade off --> in data structure called graph, in graph data structure we specify the requirements of the graph by making a adjacency matrix as shown in the figure below that contain nodes the links of each node, if the matrix is Dense(have many links in the matrix) it will be okay in memory and time but if the matrix is Sparse(few links in the matrix) it will take a lot of unused memory but with a good time so in general --> The graph data structure takes a few time to execute but with a big memory (space).



The space complexity will be (v^2), but if we want to know if the node a is connected with other nodes it takes time O(1) but finding the adjacent nodes will take 0(v) but in general some of the operations in graph will takes O(1) --> good time but the space will be high.

3) The big example of trade off -> list and linked list, in list if we want to double the size of the array,the time complexity will be O(n) but in linked list the time will be O(1), so in linked list the time will be better than the list.

4) The last example : the two algorithms-> merge sort and selection sort:

Compression	Selection sort	Merge sort
Time complexity	O(n^2)--> worst	O(nlogn)--> better
Space complexity	O(1)--> better	O(n)--> worst

So as shown in the table, the time complexity of selection sort is worst than merge but better in space complexity, and merge sort is better in time complexity than selection sort but worst in space complexity than selection sort.**(Time-Space Trade-Off in Algorithms - GeeksforGeeks, 2022)**

INDEPENDENT DATA STRUCTURE

First before talking about the benefits of implementation independent data structure, we should know what it means , independent data structure is simply means to separate two stages of the data structure which are the abstract data type stage and implementation stage for many reason such as:

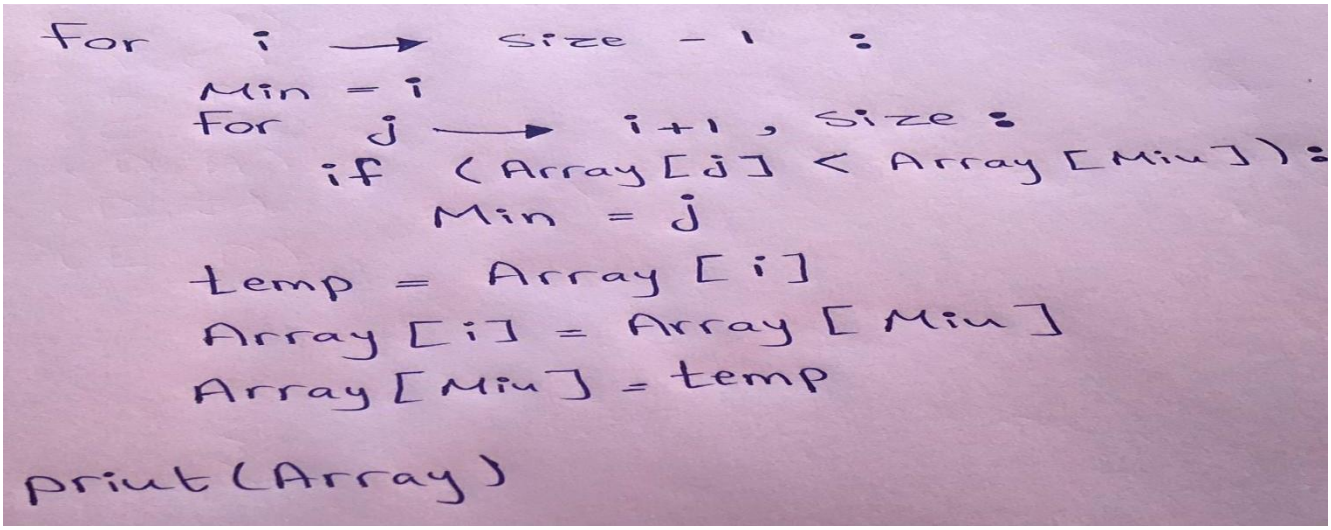
- ✧ It helps the developer to build his own requirements, and also can maintain the basic operation of the data structure. For example : in stack data structure if the developer know all about the stack and how it's work from ADT, he can implement it with his requirements such as -> choose one option between the list and linked list.
- ✧ Become more efficient in using the memory (space), for example : if the size of data is unknown then the developer will use the linked list so from the last example, if the developer has the efficient in use the data structure, he can decrease the waste of memory.
- ✧ The developer can also benefits from the implementation of particular data structure by reused it in another places or implementation another data structure or make a libraries to clients that they can use it when they need.
- ✧ Also it gives the modularity.
- ✧ Final benefit which is the program became independent of ADT representation, so we can improved the representation without breaking the program.

(Answer in Java | JSP | JSF for Rakan #149033, 2022)

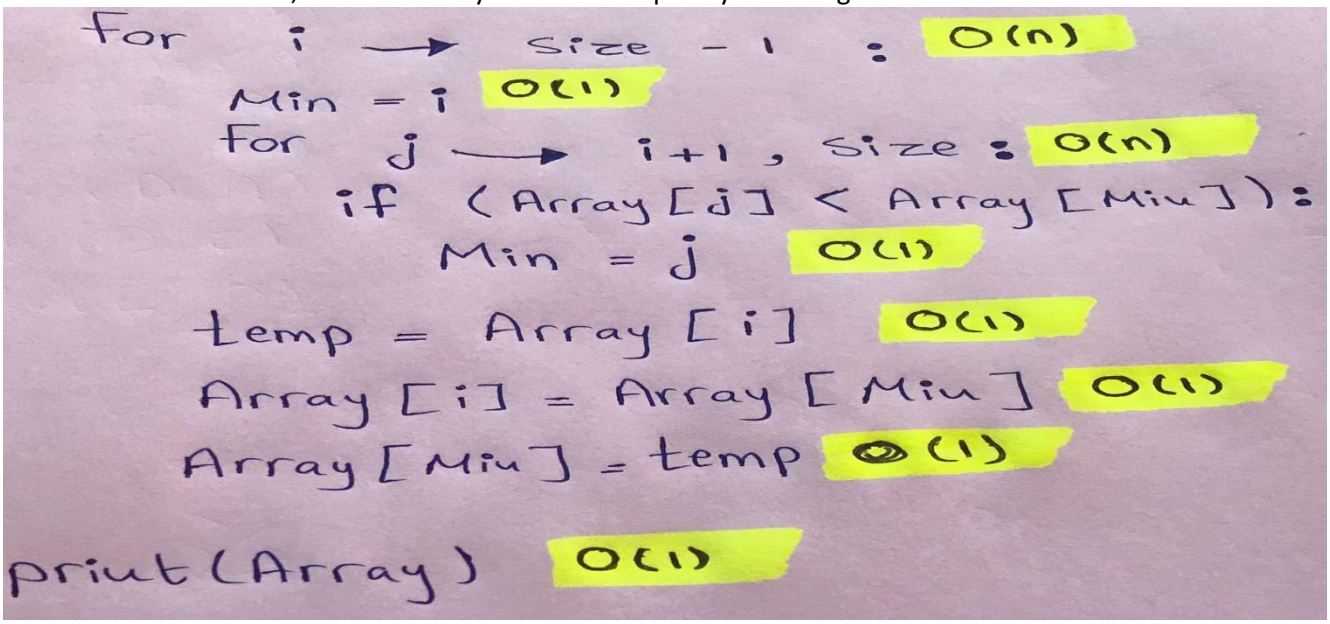
Compare between selection sort and merge sort

Selection sort algorithm is type of sorting algorithms that sort the element from small element to large element or from large to small element in the array. In selection sort we will scan all the elements and find the small elements and swap it with the large elements to sort.

✧ Scudo code of the algorithm:



So from the scudo code , we can identify the time complexity of the algorithm as shown below:

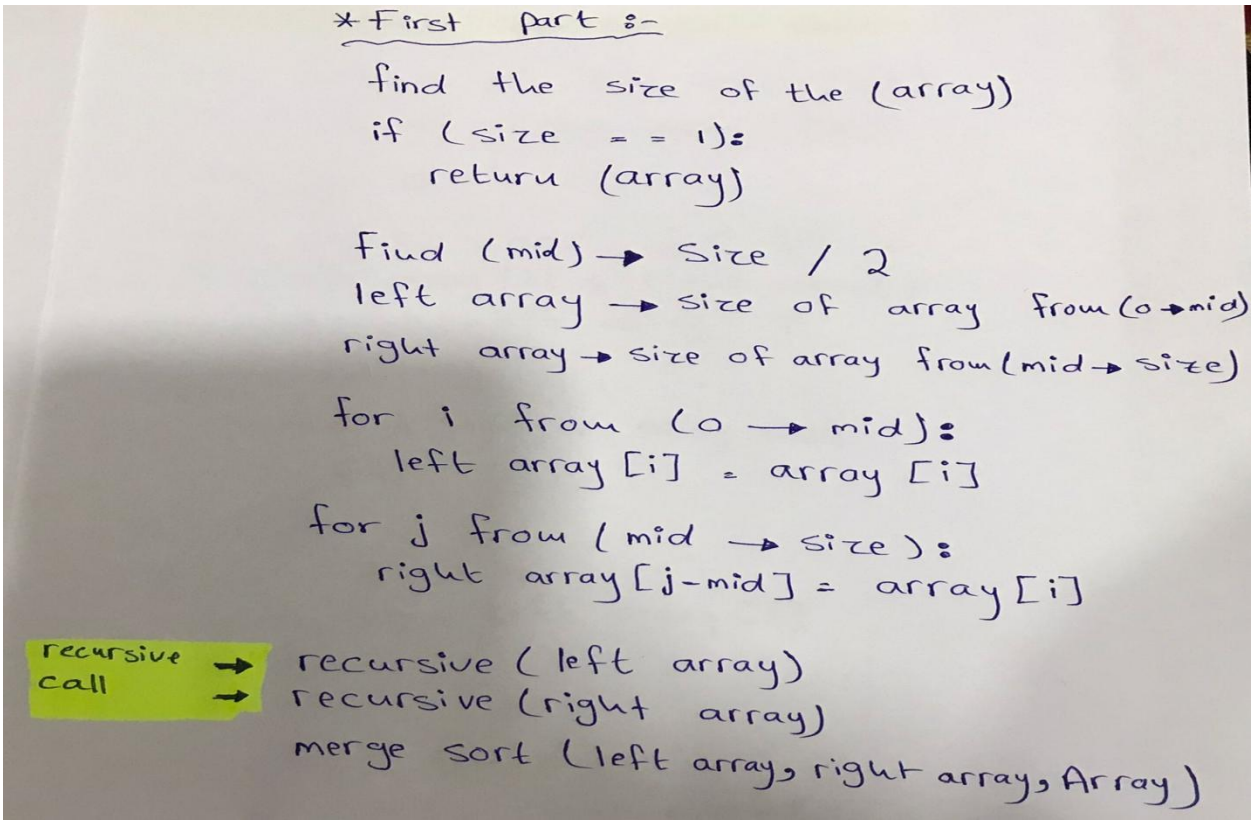


And we all know to calculate the time complexity , we should take the worst case(big time in the code):

T = O(1)+O(n^2) -> T= c6+O(n^2)-->T = O(n^2)
Time complexity of Selection sort --> O(n^2)

Another type of sorting algorithms is the **merge sort** -> use the divide and conquer strategy in this algorithm and works by dividing the list into equals size of two array and sort them and in the final step combine the elements again (with sorted elements).

- Scudo code of the algorithm:



* Secoud part ⇒ merge sort

fiud the size of left array (SL)
find size of right array (SR)

```
i = K = j = 0

While i < SL and j < SR :-
  if (left array [i] ≤ right array [j]:
      array [K] = left array [i]
      i = i+1
  else:
      array [K] = right array [j]
      j = j+1

K = K+1
While ( i < SL):
    array [K] = left array [i]
    j = j+1
    K = K+1
while ( j < SR):
    array [K] = right array [j]
    j = j+1
    k = K+1
```

From the pseudo code above, we can identify the time complexity of the algorithm. If you noticed that the array is dividing into equal arrays and that means -> array size (n) is divided by (logn) times.

logn is calculated because we reduced the size of the array by calculate the mid as shown in the figure: The first size -> n ,Second-> n/2, Third-> n/4 ---- .So the equation -> $n/2^k$ until we reached (1) ---> $1 = n/2^k$, and mathematically to find the value of k we will get the logn. (This paragraph to how we get the logn)

The merge processes (combine the element again into one array) it will take O(n) time complexity because there is while loop.So the final time complexity:

T = O(nlogn)

✧ Compare performance between two algorithms:

Compression	Selection sort	Merge sort
Time complexity	O(n^2)--> worst	O(nlogn)--> better
Space complexity	O(1)--> better	O(n)--> worst

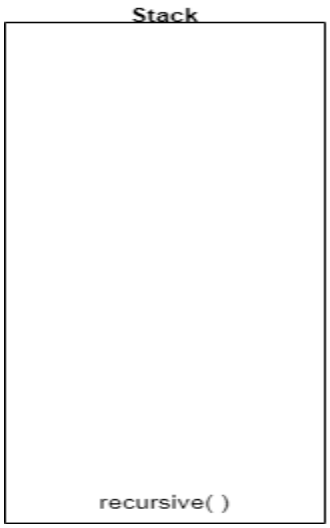
In my opinion, I will use the merge sort because it gives the better time complexity and it will show the output faster because if we use a large input in selection sort ,it maybe needs a lot of time to execute to show the output because of O(n^2) instead of merge with time (nlogn) but it will take a lot of space.

***Note**

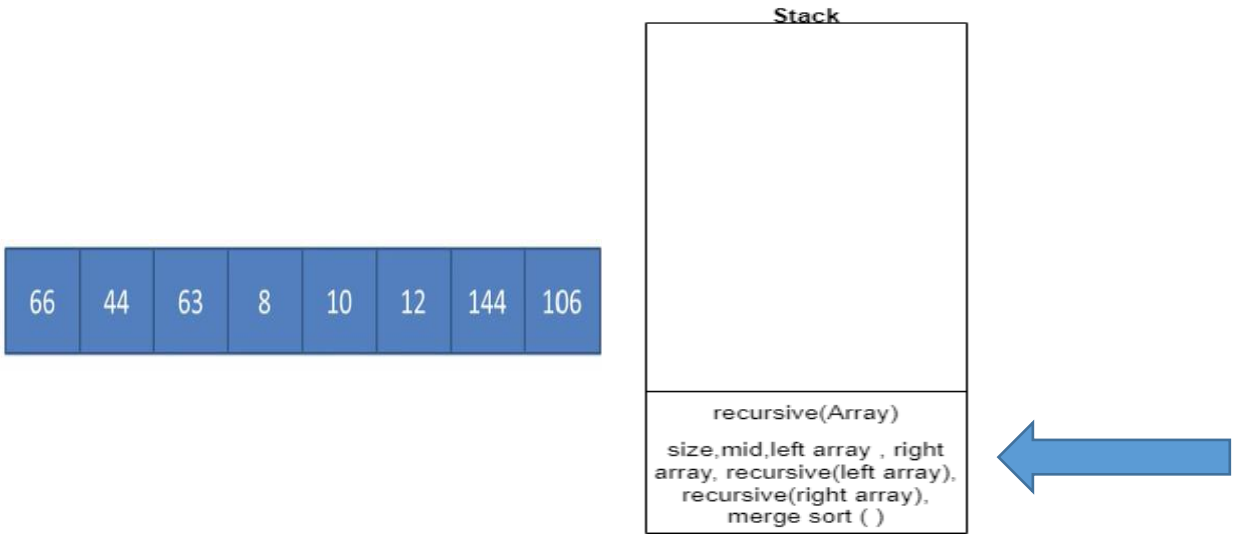
These two algorithms can mention in the trade off because one is better in time complexity and the other better in space complexity.

CALL STACK

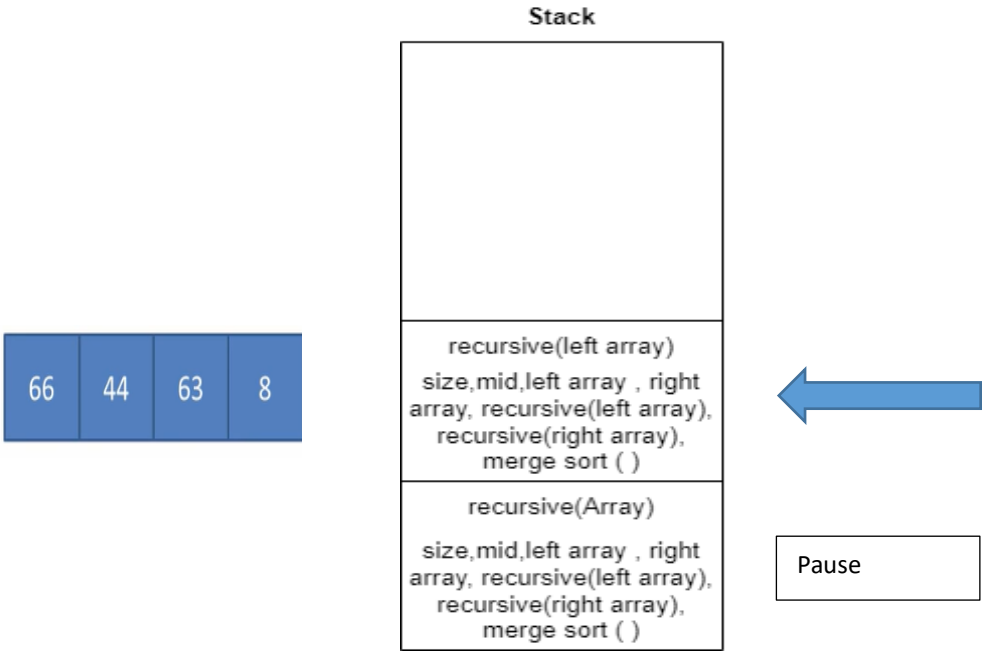
From the pseudo code before, we call **identify the operation of memory stack and how it used to implement function calls ans suppose we have this array [66,44,63,8,10,12,144,106] as example:**



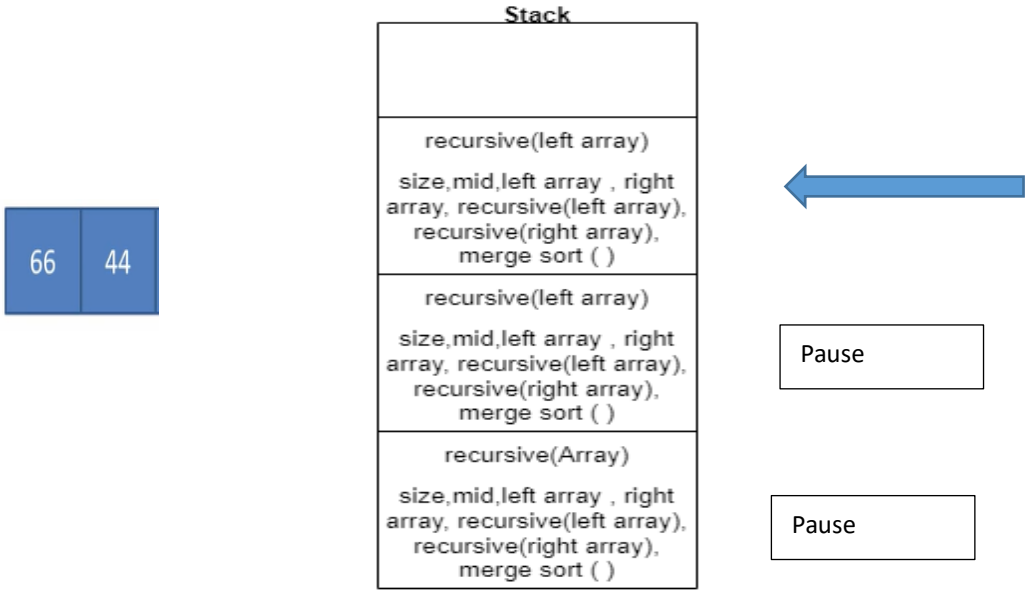
As shown in the figure, there is a table that contains the memory reserved as stack segments from the pseudo code. When we run the code, the first function (recursive) invoked so the recursive function pushed on the stack.Note that -> during the run, the function at the top of the stack is always executing and the below of it will pause until the top finished and become the top of the stack.



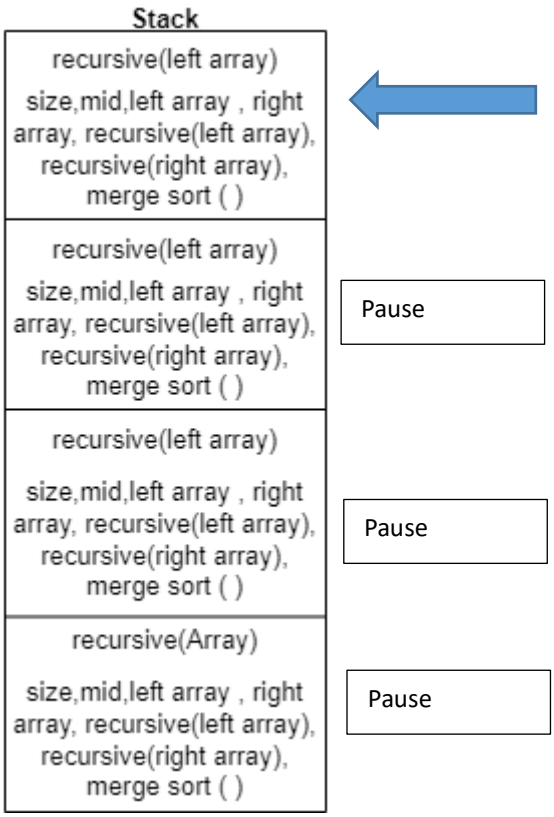
The recursive function with (Array is the parameter of the function) and variables and functions must store in the stack.So in this function there are many variables and all stored in the stack as shown in the figure.



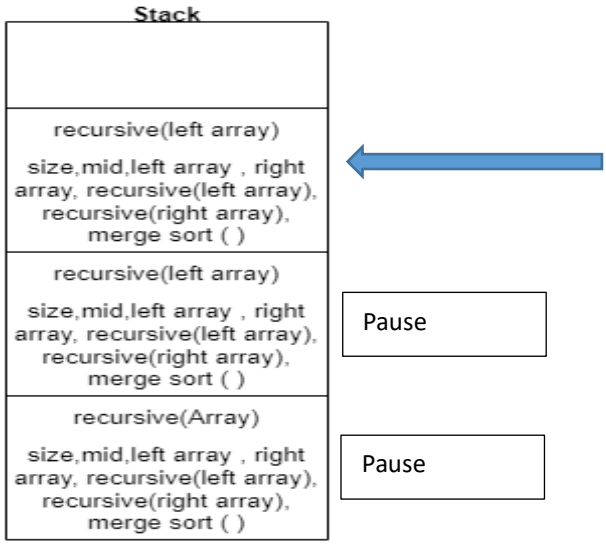
From the code, the recursive function calls it self but with different parameter (left array), and it also has all the variables so it will push to the stack. And note that the recursive (left array) is executing because it's the top of the stack while recursive(Array) is pausing until becoming the top of the stack.



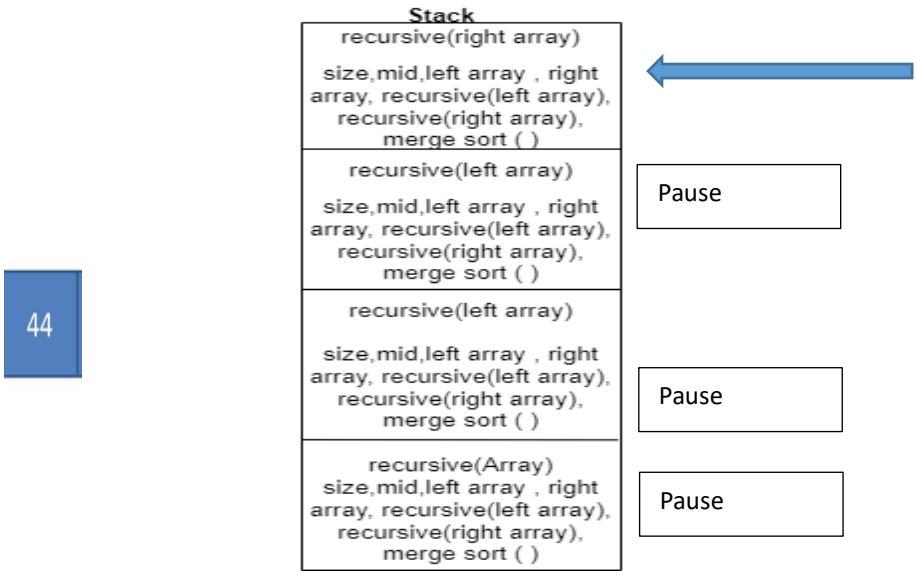
Also the recursive function with parameter left array calls it self but also with different value of parameter (left array) so it will push to the stack, and this step will repeat until enter the (if clause) which is become the size of the array less than (2) but if size doesn't reach the number that less than number (2) -> it will repeat and call the function recursive (left array) until reach the number that less than number (2).Note that the the function at the top is executing because it is the top and the others are pausing.



the top of the stack -> the size of left array is less than number (2) so from the pseudo code the if clause will entered and return the Array as shown in the pseudo code.

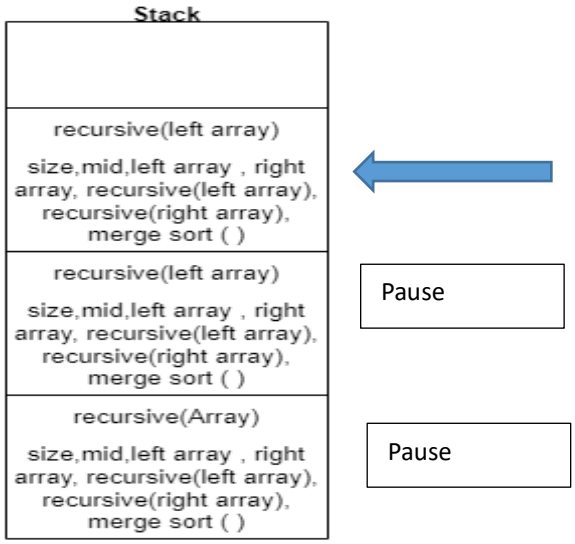


So after return Array the function will end,and it will pop from the stack and the top of the stack will resume as shown in the figure.

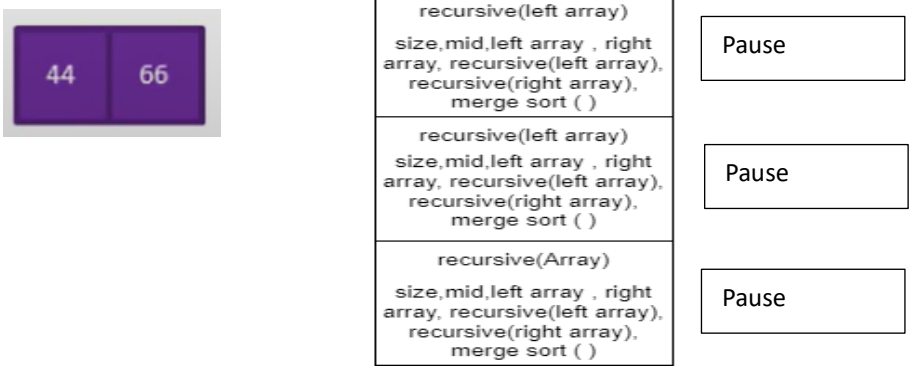


After resuming the top of the stack (second recursive (left array)), it will continue run the code and call again the recursive function but with different parameter which is right array and stored all the variables in the stack so it will push to the stack.

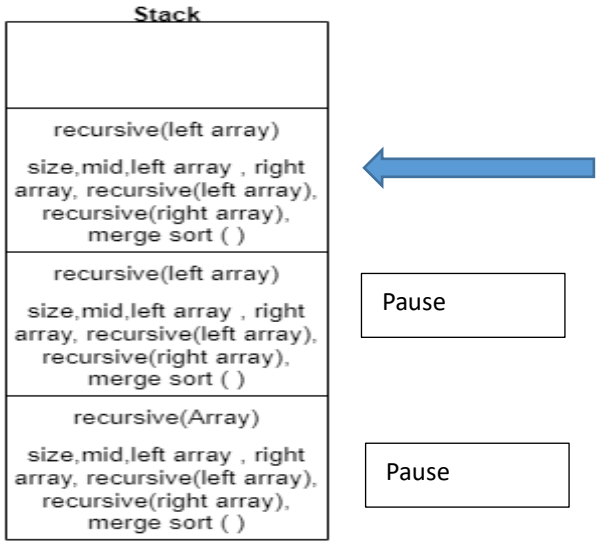
Note In case (merge sort) -> from the step before ,if the left array is less than number two, that means that the array can't split it self again and usually it contains one element on the left and one element on the right as shown below:



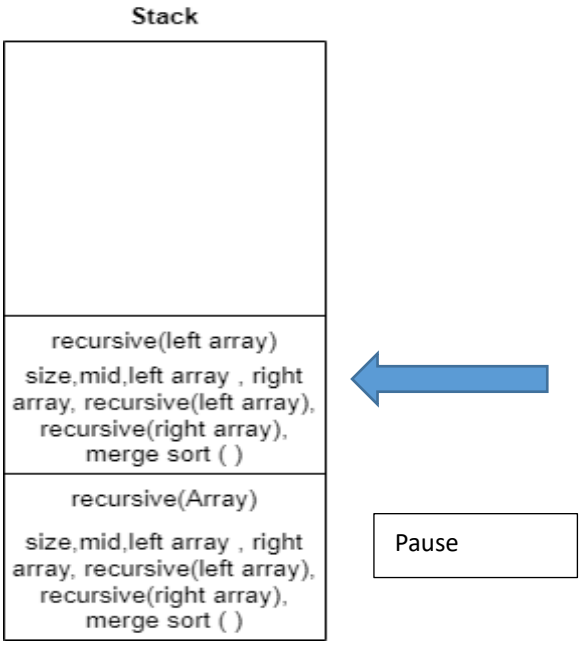
From the note before, the function recursive with (right array parameter) will be less than number (2) so it will return array and after returning the array , it will pop from the stack and make the second recursive function (left array) the top of the stack and continue run the code.

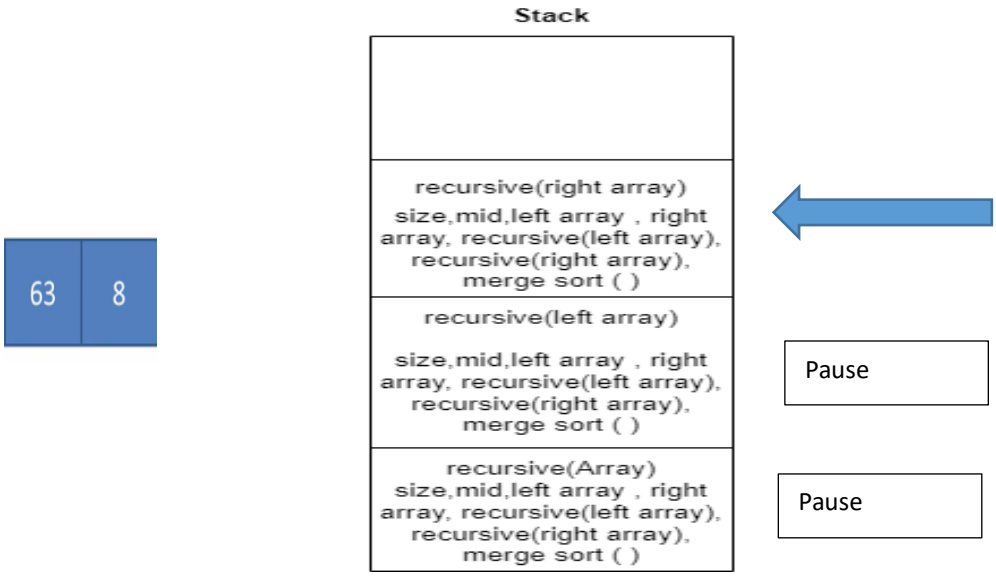


While continue run the second recursive (left array)-> it will call the function merge sort that has the parameters (66,44,Array) and do the work of the function so it will push to the stack.

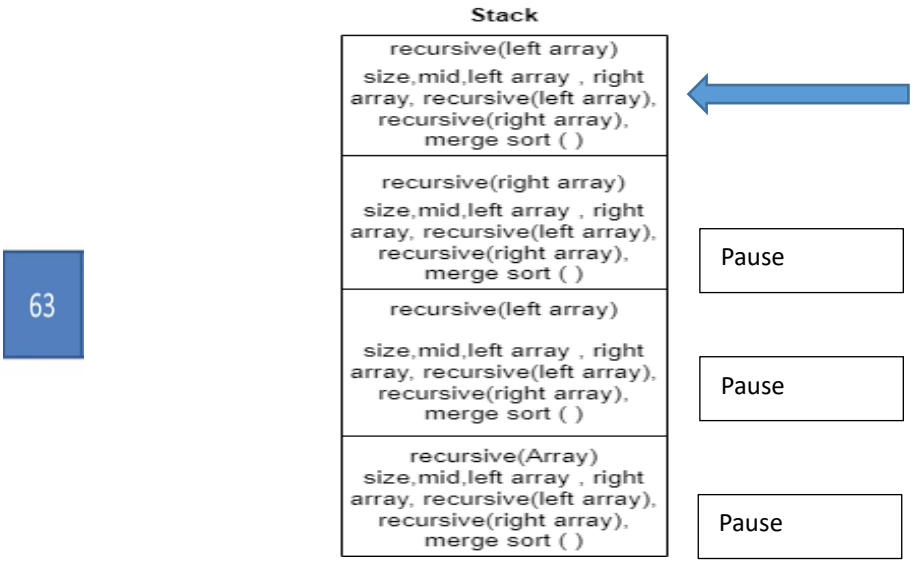


After doing the work of the function merge sort, it will pop from the stack and continue run the second recursive (left array) , but as shown in the pseudo code-> the second recursive (left array) is finished because it runs all the code from the beginning to the end so it will pop from the stack and resume the first recursive (left array) as shown in the figure below:

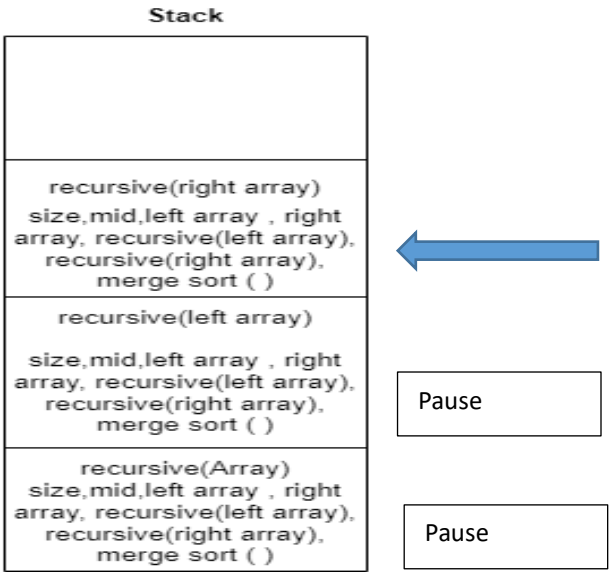




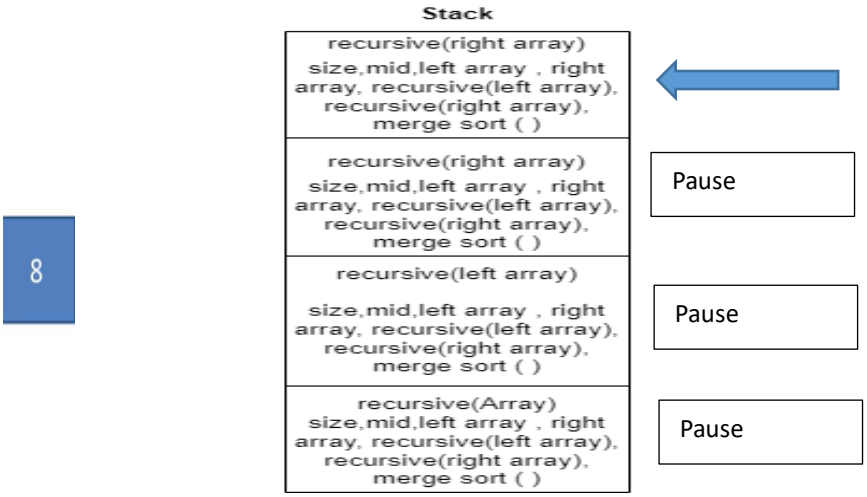
After resuming the recursive(left array), it will called the function recursive (right array) and it will pushed in the stack, the top will run and recursive(left array) pause.



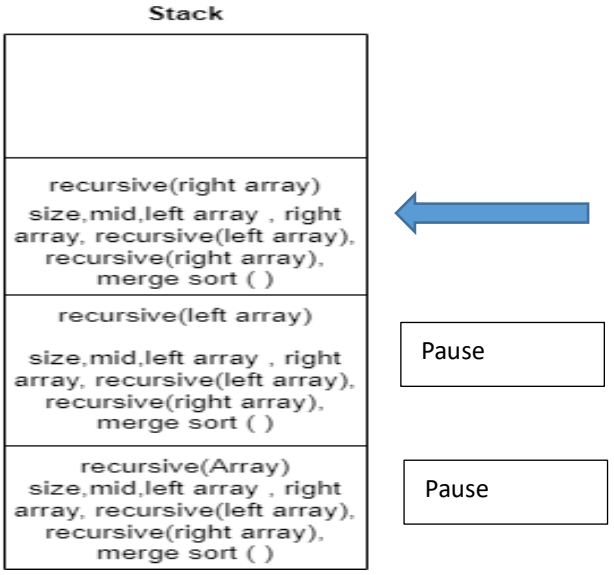
In recursive (right array), it will run and also call the function recursive (left array) so it will push in the stack as shown in the figure. And from the previous note, recursive function will repeat call it self until reach number that is less than number (2), and the size of left array is less than number two , so it will enter the if clause and return Array.



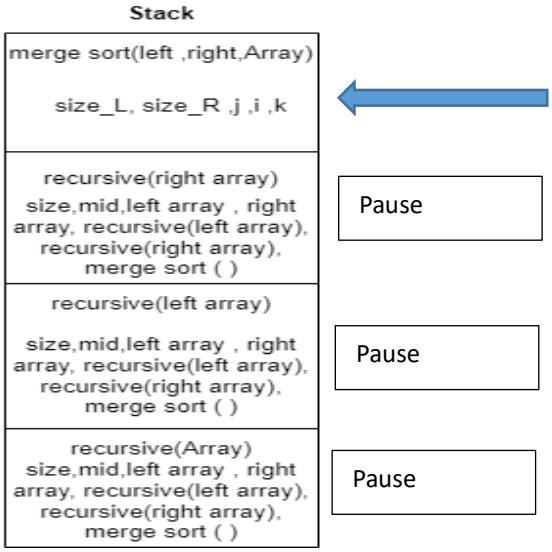
After return Array, it will pop from the stack and resume top of the stack which is recursive(right array).



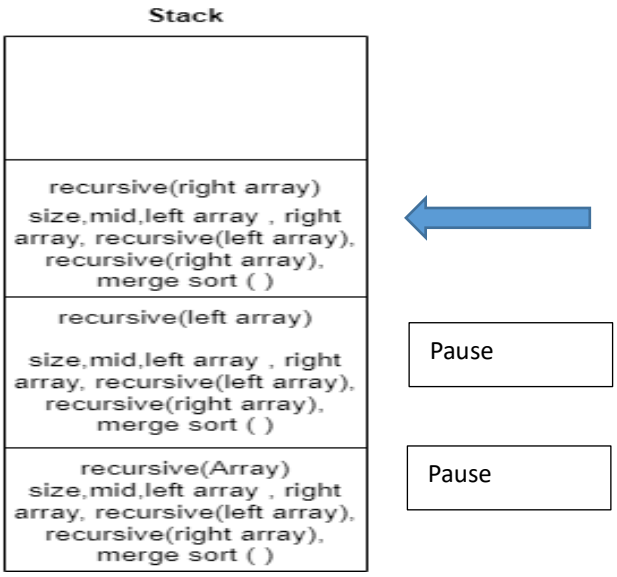
After resuming the recursive (right array) and continue run the code, it will called the function recursive(right array) so it will push to the stack.



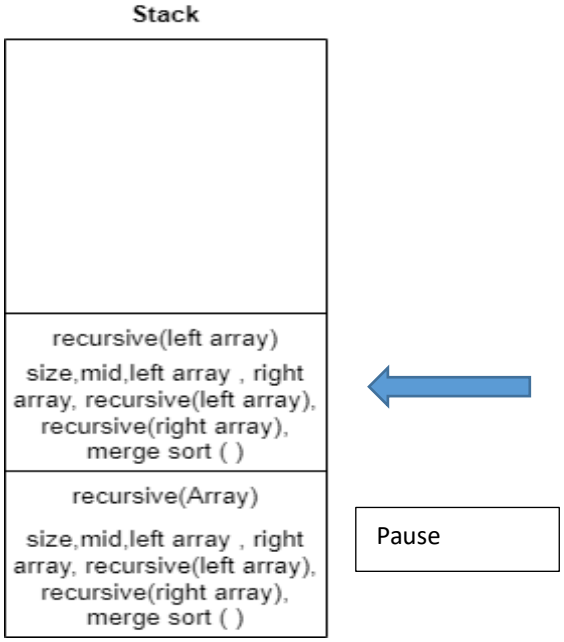
From the note before, if the size of recursive(left array) is less than number two , that means that the array can't split and usually it reached one element in the left array and one element in the right array , so in the second recursive(right array) -> the size of the array is less than two -> it will enter the if clause and return Array and finished. After finish it will pop from the stack and the recursive (right array) will resume.



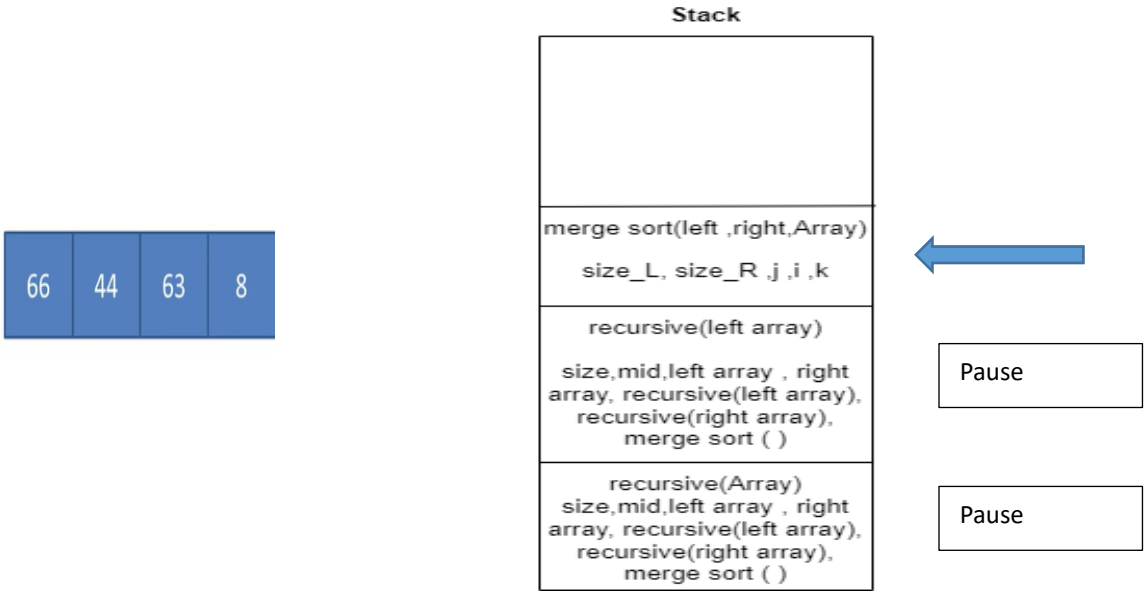
While resuming the recursive (right array) and continue the code, it will call the function merge sort with the parameters (63 , 8 , Array) so it will push to the stack and store all the variables in the stack.



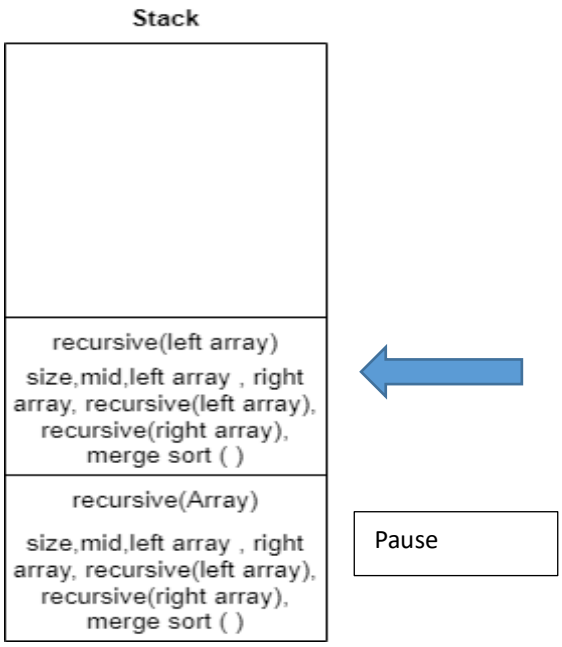
After doing the work of the function merge sort, it will pop from the stack and the recursive (right array) will resume.



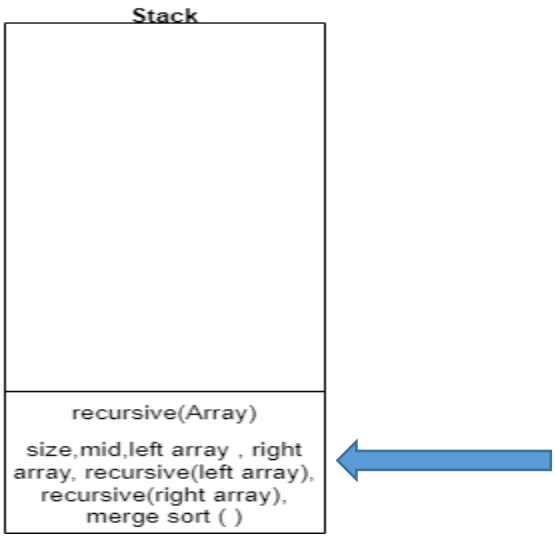
After resuming the function , the code finish so it will pop from the stack and recursive (left array) will resume.



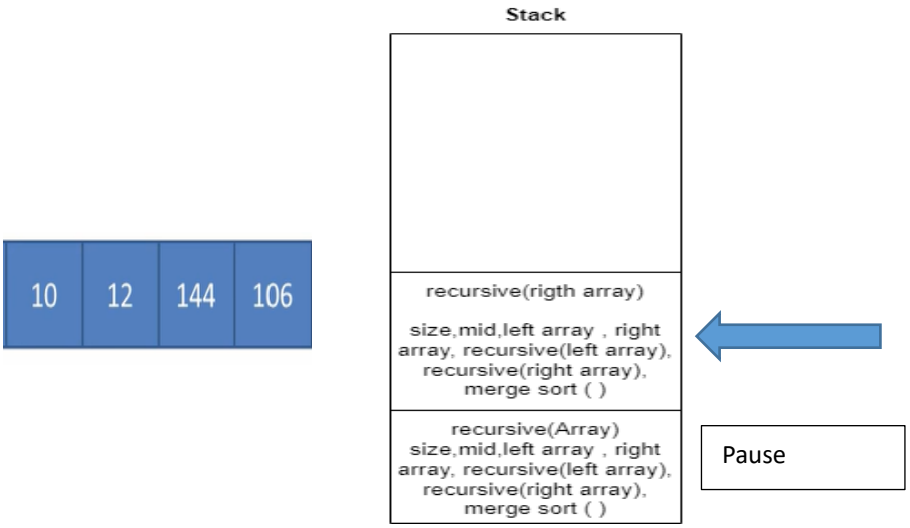
While back to recursive(left array) and continue run the code, it will call the function merge sort with the parameters ([44,66] , [8,63] , Array) so it will push to the stack and stored all the variables.



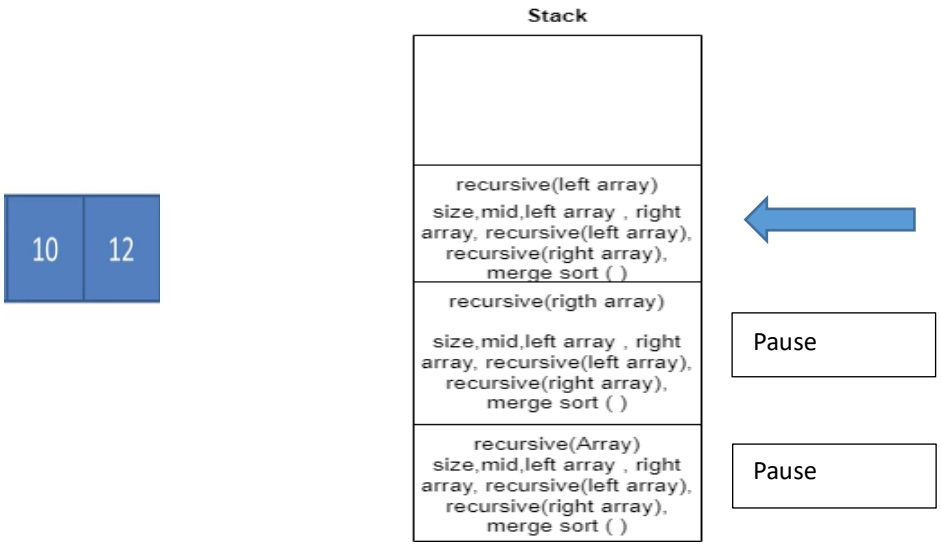
After done the work of the function merge sort, it will pop from the stack and resume the recursive (left array).



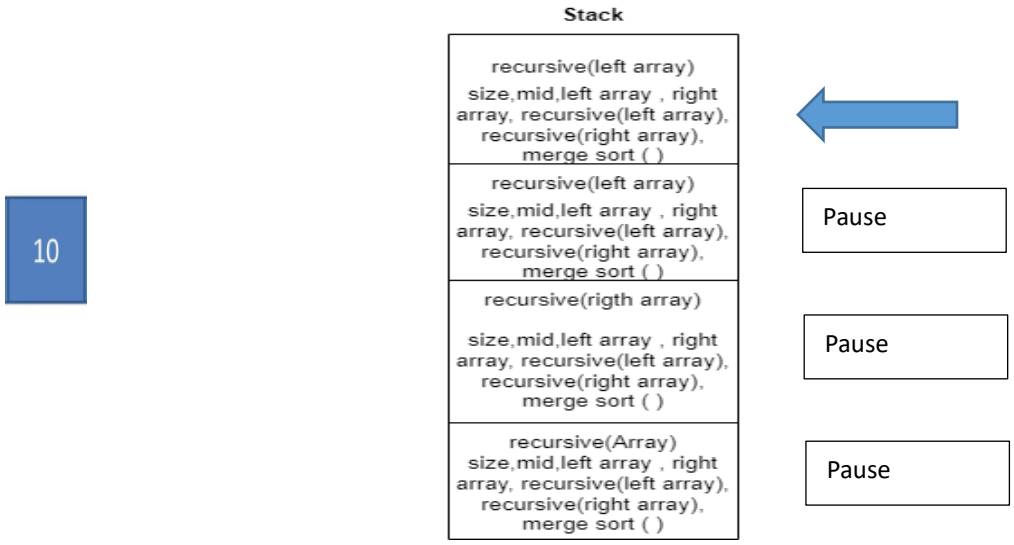
While resuming the code of the recursive (left array), the code finished so it will pop from the stack and recursive(Array) will resume.



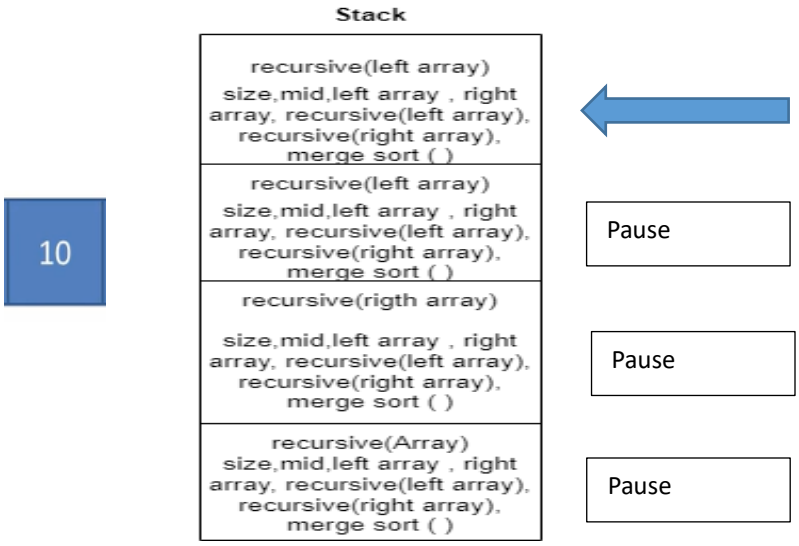
recursive(Array) will continue and run the code, and called the function recursive(right array) so it will push to the stack and run.



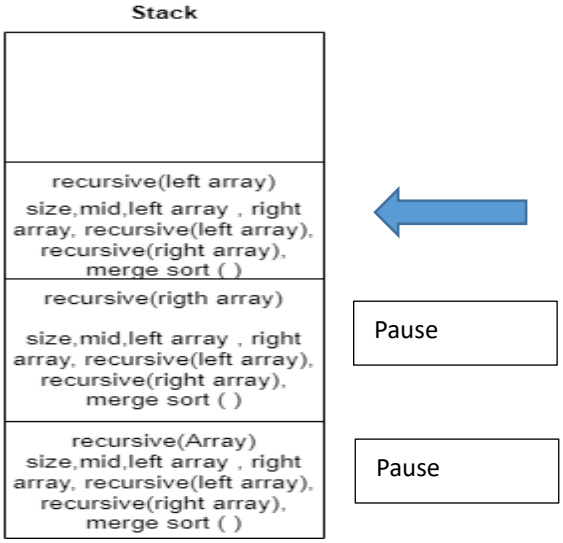
While resuming the recursive (right array), it will call the function recursive(left array) so it will push to the stack and run.



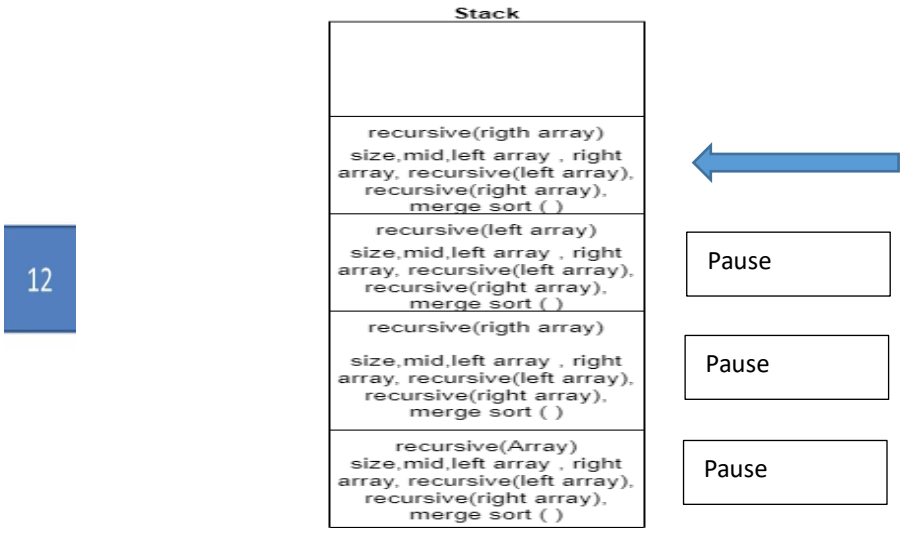
Also the recursive function with parameter left array calls it self but also with different value of parameter (left array), and this step will repeat until enter the (if clause) which is become the size less than (2) but if size doesn't reach the number that less than number (2) -> it will repeat and call the function recursive (left array) until reach the number that less than number (2).Note that the the function at the top is executing because it is the top and the others are pausing.



the the top of the stack -> the size of left array is less than number (2) so the if clause will entered and return the Array as shown in the pseudo code.

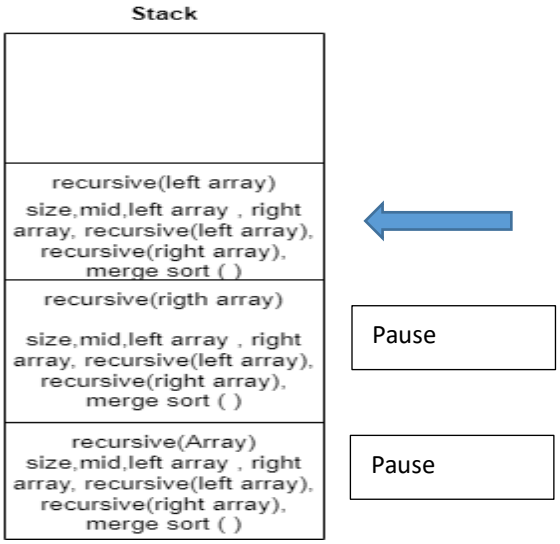


So after return Array , it will pop from the stack and the top of the stack will resume.

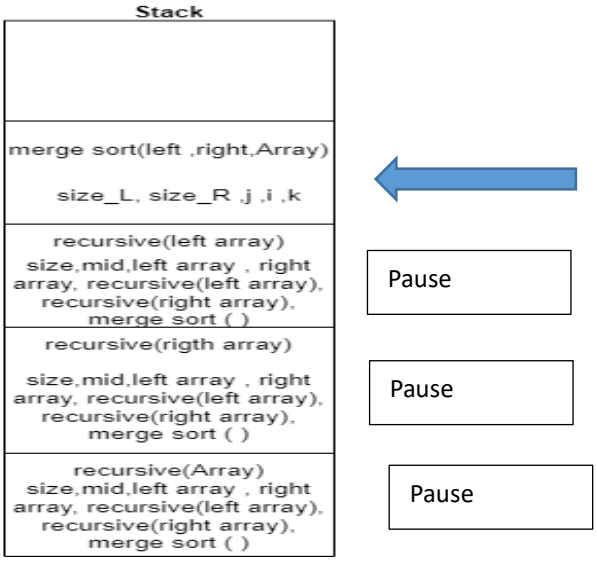


After resuming the top of the stack (second recursive (left array)), it will continue run the code and call again the recursive function but with different parameter which is right array and stored all the variables in the stack.

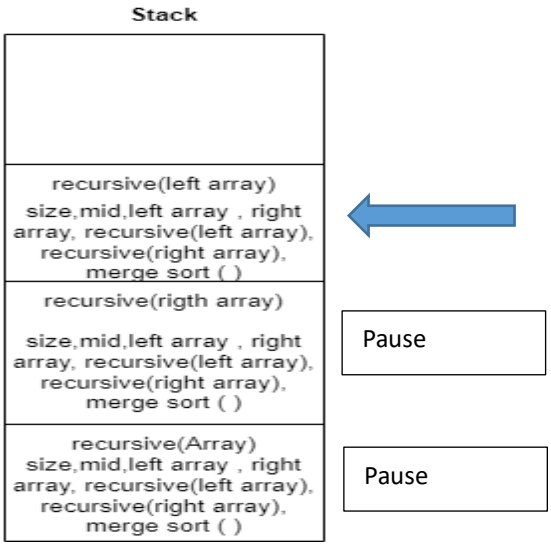
Note In case (merge sort) -> from the step before ,if the left array is less than number two, that means that the array can't split it self again and usually it contains one element on the left and one element on the right.



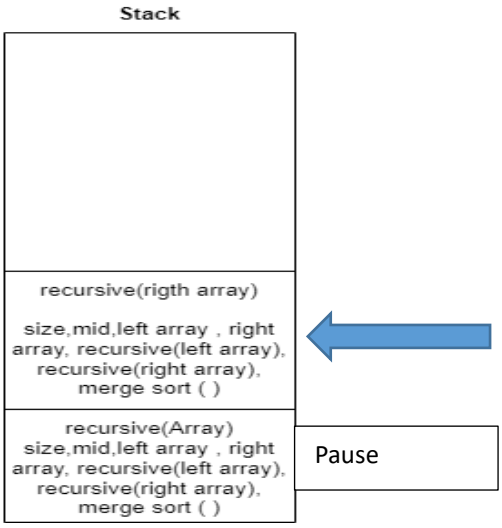
So from the previous note, the recursive(right array) will enter the if clause and return Array because the size of the array is less than 2 and finish the function. After finish the function , it will pop from the stack and resume the recursive(left array).



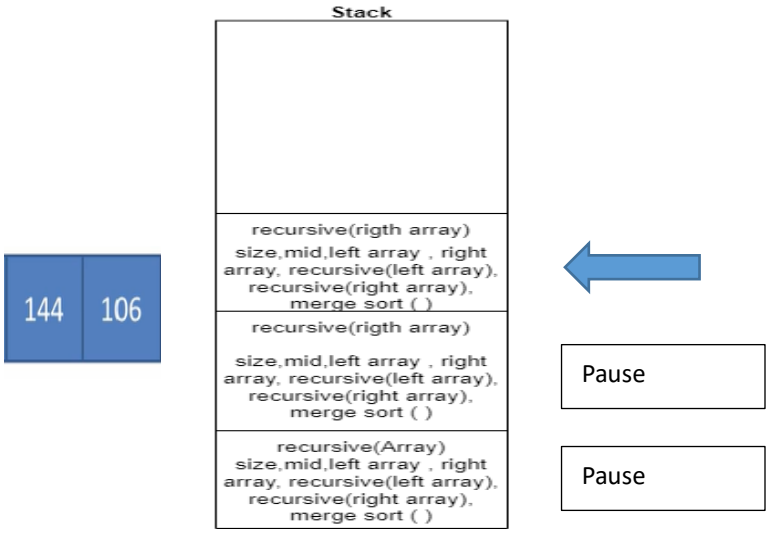
After resuming the recursive(left array) and continue run the code, it will call the function merge sort with the parameters(10, 12 , Array) so it will push to the stack and run it.



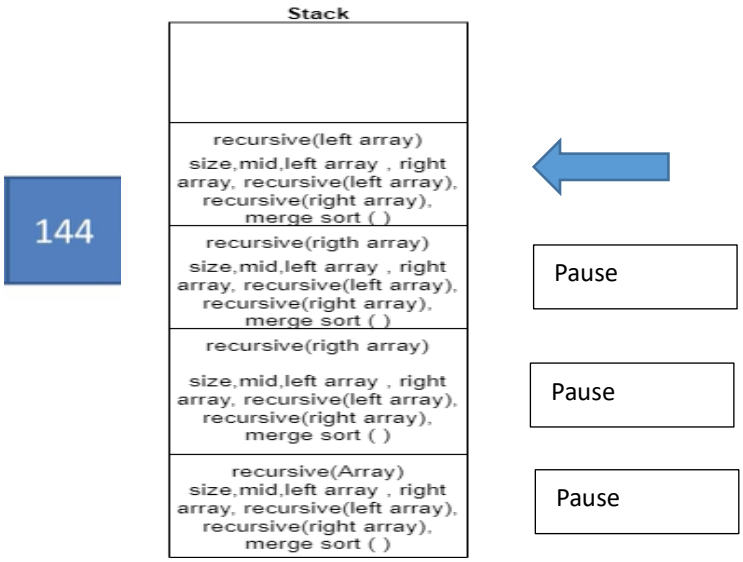
After do the work of the function merge sort, it will pop from the stack and the recursive(left array) will resume.



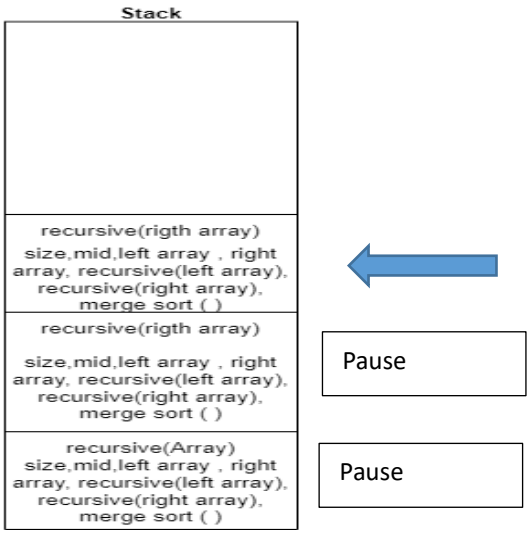
After resume the recursive(left array) , the function finished so it will pop from the stack and recursive(right array) will resume.



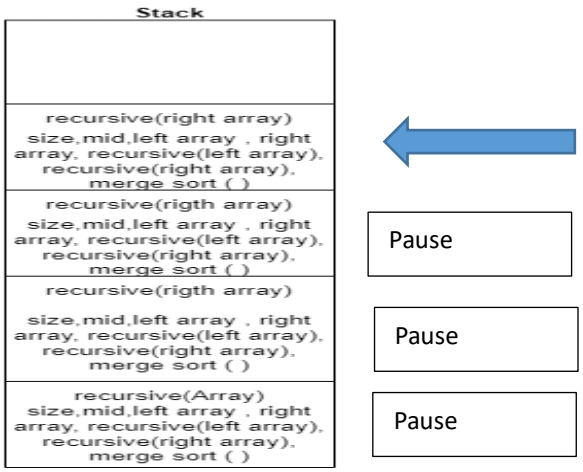
Resuming the recursive(right array) and continue run the code, it will call the function recursive(right array) so it will push to the stack.



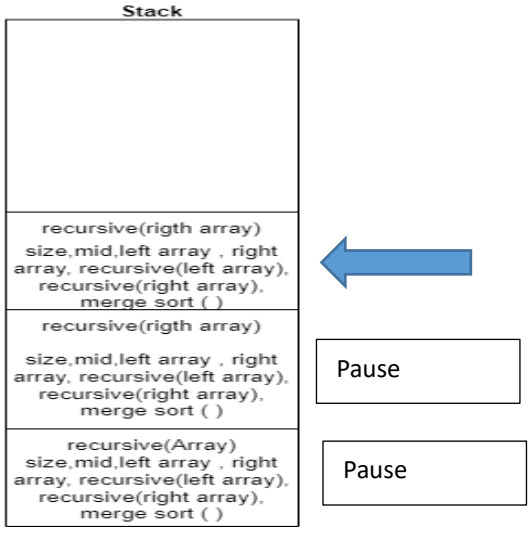
In second recursive (right array), it will call the function recursive (left array) so it will push to the stack and run it .



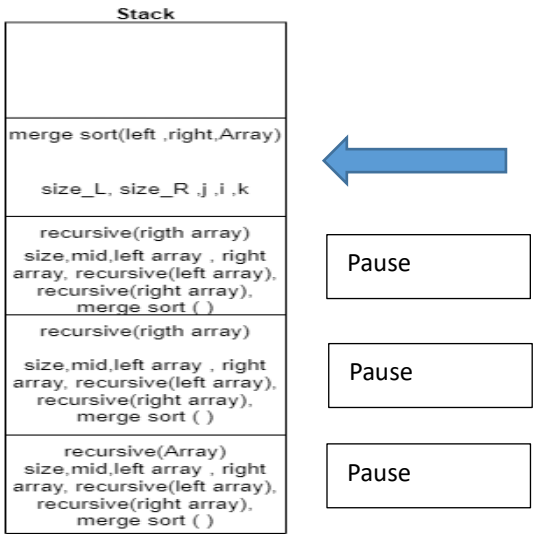
that the size of the array (left array) is less than two, so it will enter the if clause and return Array, after that it will finish the function, so it will pop from the stack and resume recursive(right array).



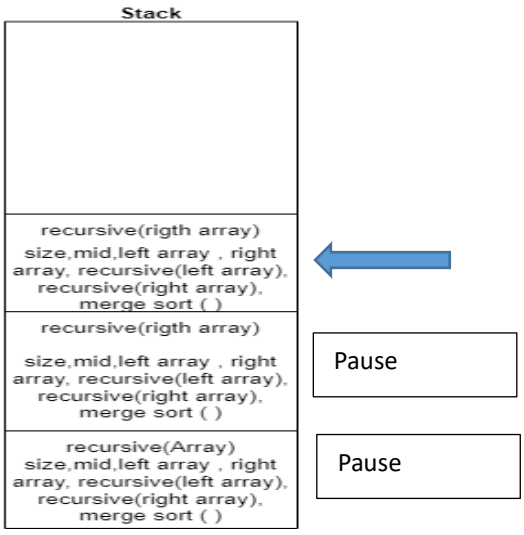
While resuming the recursive(right element) and continue the code,it will call the function recursive (right element) so it will push to the stack and run it.



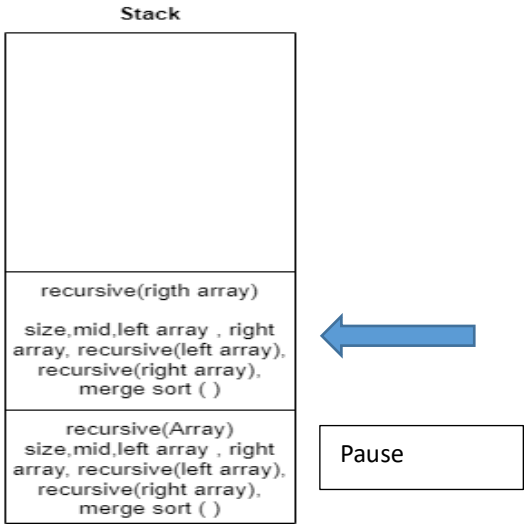
From the previous note, if the size of the left array is less than 2-> it means that array cant split it again and usually the left array is contains one element and right array one element.And before , the size of left array is less than 2 so the right array will also be less than two so it will enter the if clause and return Array and finish the function so it will pop from the stack.



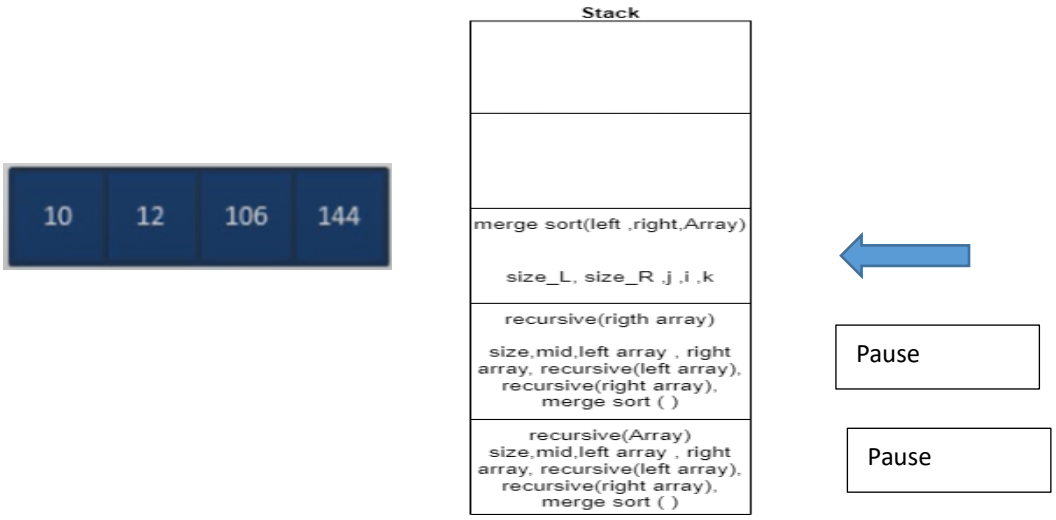
While resuming the recursive(right array) can continue run the code, it will call function merge sort (144,106,Array) so it will push to the stack.



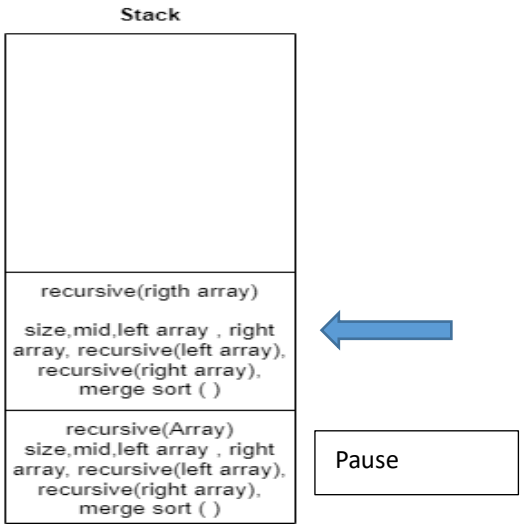
After do the work of the function merge, it will pop from the stack and resume the recursive (right array).



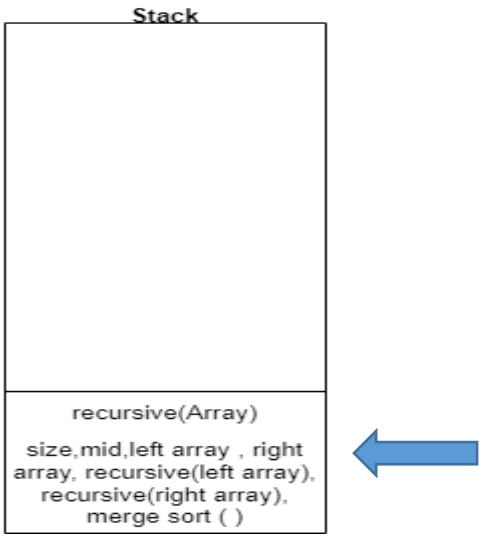
After finish the function merge sort, the recursive (right array) will continue run the code , but the code finished so it will pop from the stack and first recursive(right array) will resume.



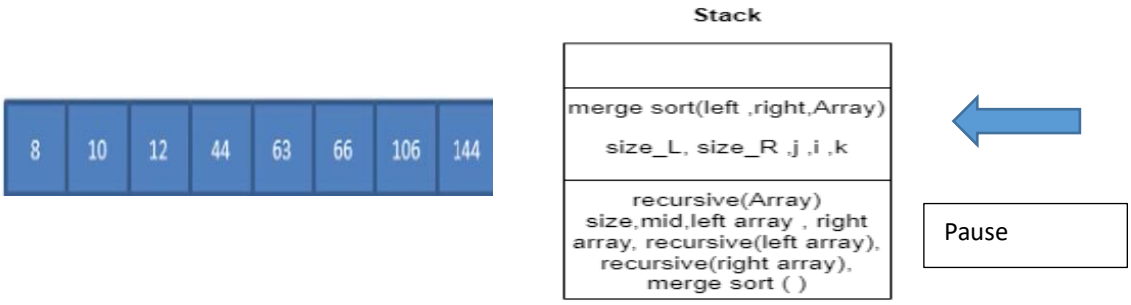
While resuming the recursive(right array) can continue run the code, it will call function merge sort ([10,12],[106,144],Array)so it will push to the stack.



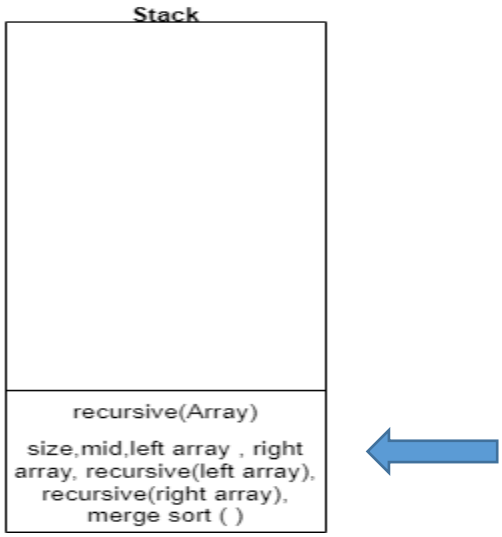
After do the work of the function merge, it will pop from the stack and resume the recursive (right array).



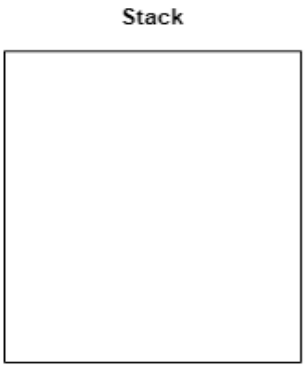
After finish the function merge sort, the recursive (right array) will continue run the code , but the code finished so it will pop from the stack and recursive (Array) will resume.



Resume the recursive(Array) it will call the function merge sort ([8,44,63,66],[10,12,106,144]) ,so it will push to the stack and run it.



After do the work of the function merge, it will pop from the stack and resume the recursive (Array).



recursive(Array) finishes run the code so it will pop from the stack and the program will also ends.

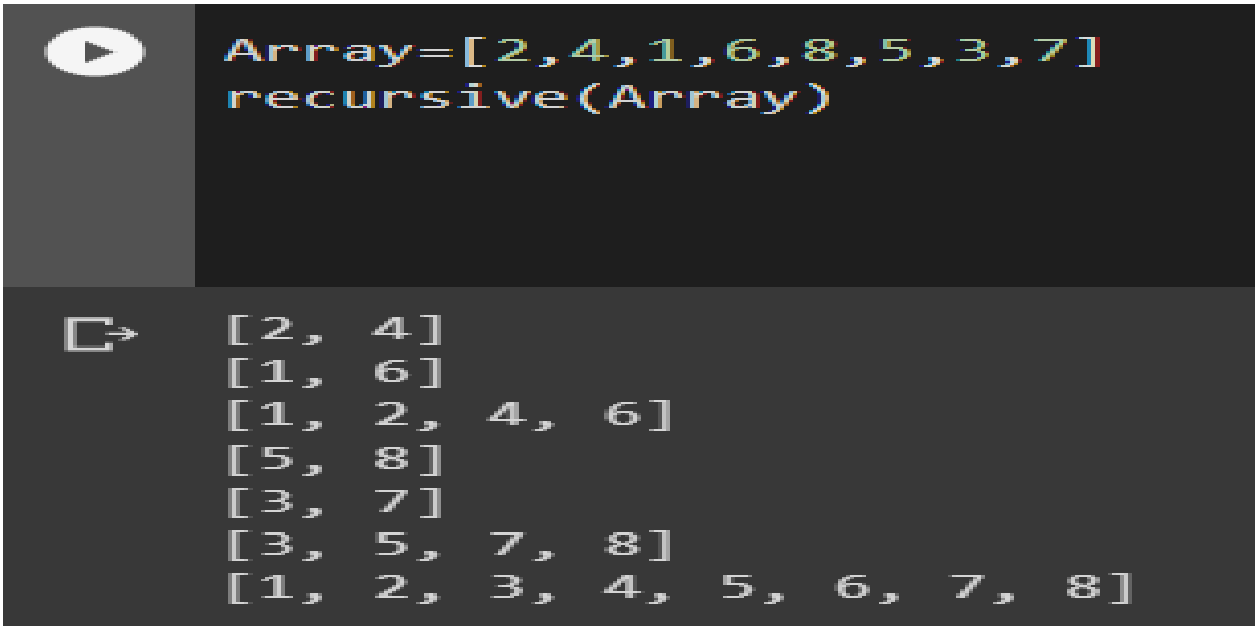
IMPLEMENTATION MERGE SORT

The implementation using python:

```
def recursive(Array):
    size=len(Array)
    if(size==1):
        return Array

    mid=size//2
    left_array=Array[:mid]
    right_array=Array[mid:size]

    for i in range(0,mid):
        left_array[i]=Array[i]
    for j in range(mid,size):
        right_array[j-mid]=Array[j]

    recursive(left_array)
    recursive(right_array)
    merge_sort(left_array,right_array,Array)
```

```
def merge_sort(left_array,right_array,Array):
    size_L=len(left_array)
    size_R=len(right_array)
    i=0
    j=0
    k=0
    while(i<size_L and j< size_R):
        if(left_array[i]<= right_array[j]):
            Array[k]=left_array[i]
            i=i+1
        else:
            Array[k]=right_array[j]
            j=j+1

        k=k+1
    while(i<size_L):
        Array[k]=left_array[i]
        i=i+1
        k=k+1
    while(j<size_R):
        Array[k]=right_array[j]
        j=j+1
        k=k+1
    print(Array)
```

Test the code:



```
Array=[2,4,1,6,8,5,3,7]
recursive(Array)
```

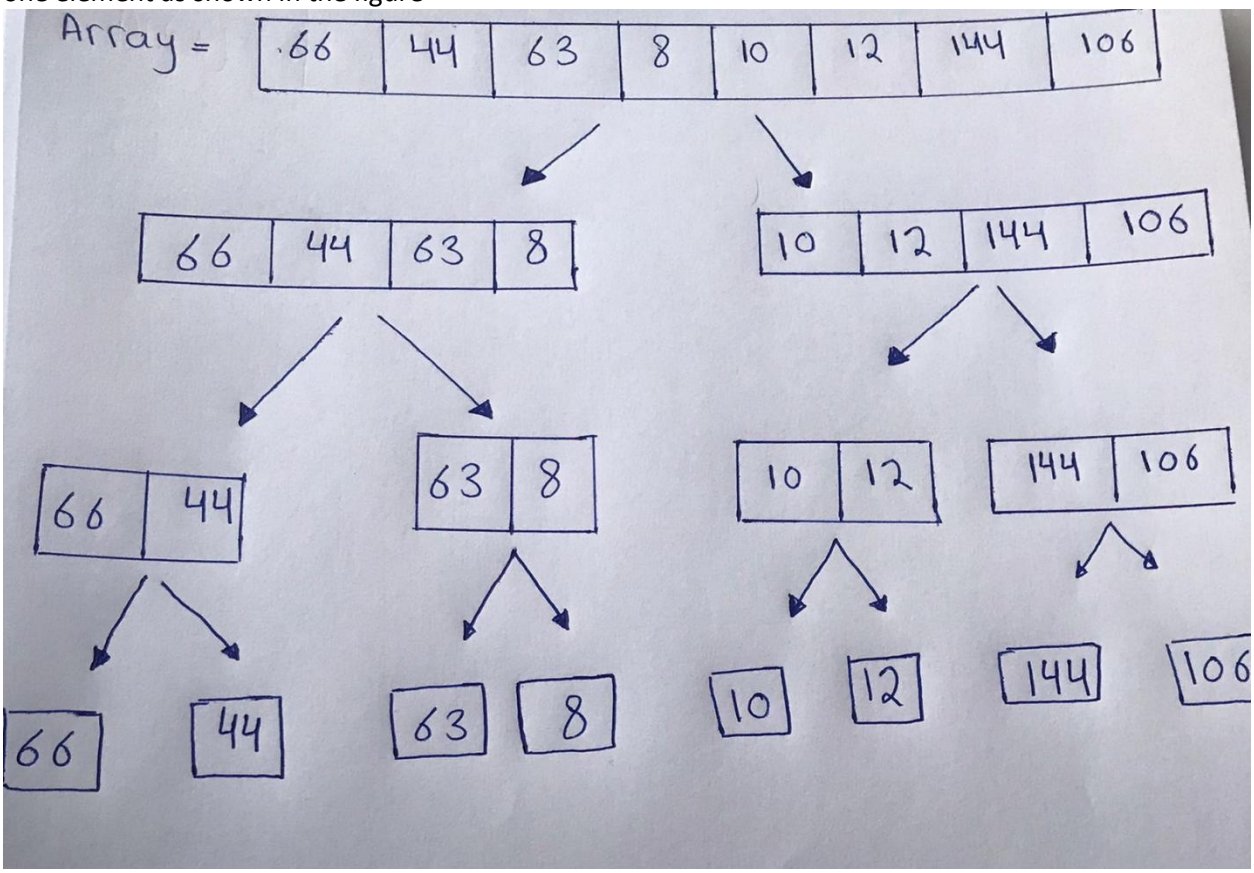
```
[2, 4]
[1, 6]
[1, 2, 4, 6]
[5, 8]
[3, 7]
[3, 5, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8]
```

ADT OF MERGE SORT ALGORITHM

After showing the implementation of the algorithm and how it deals in call stack, we must make an ADT of the algorithm and discuss more about it. In this algorithm (my program) I made two main operations :
- recursive (Array) -> formal definition
-merge_sort (left_array,right_array,Array) -> formal definition

- Recursive (Array)

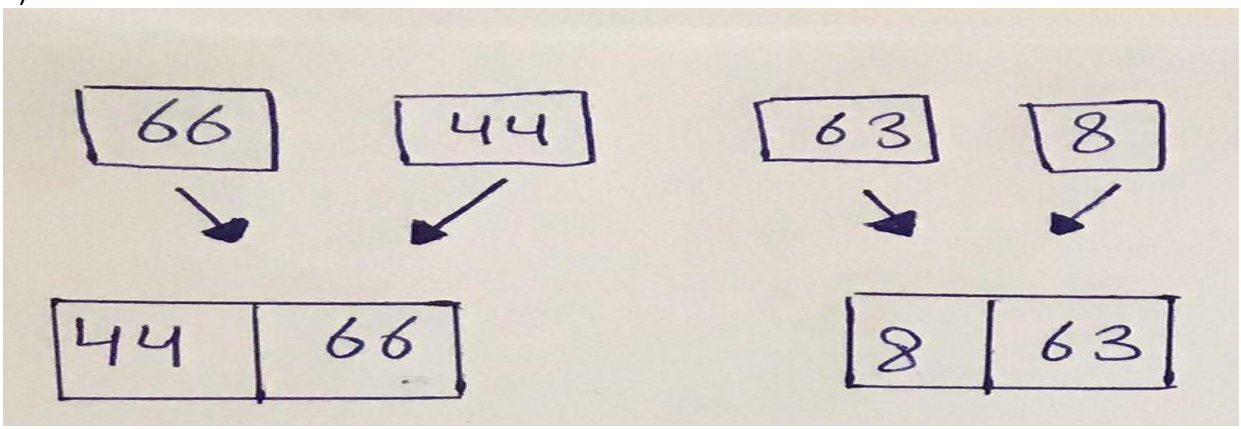
Suppose if we have this array (66 , 44, 63 , 8 , 10 , 12 , 144 ,106) and we want to implement merge sort. This operation first split the array into equal two arrays repeatedly until we reach the array that has only one element as shown in the figure



- Merge_sort (left_array,right_array,Array)

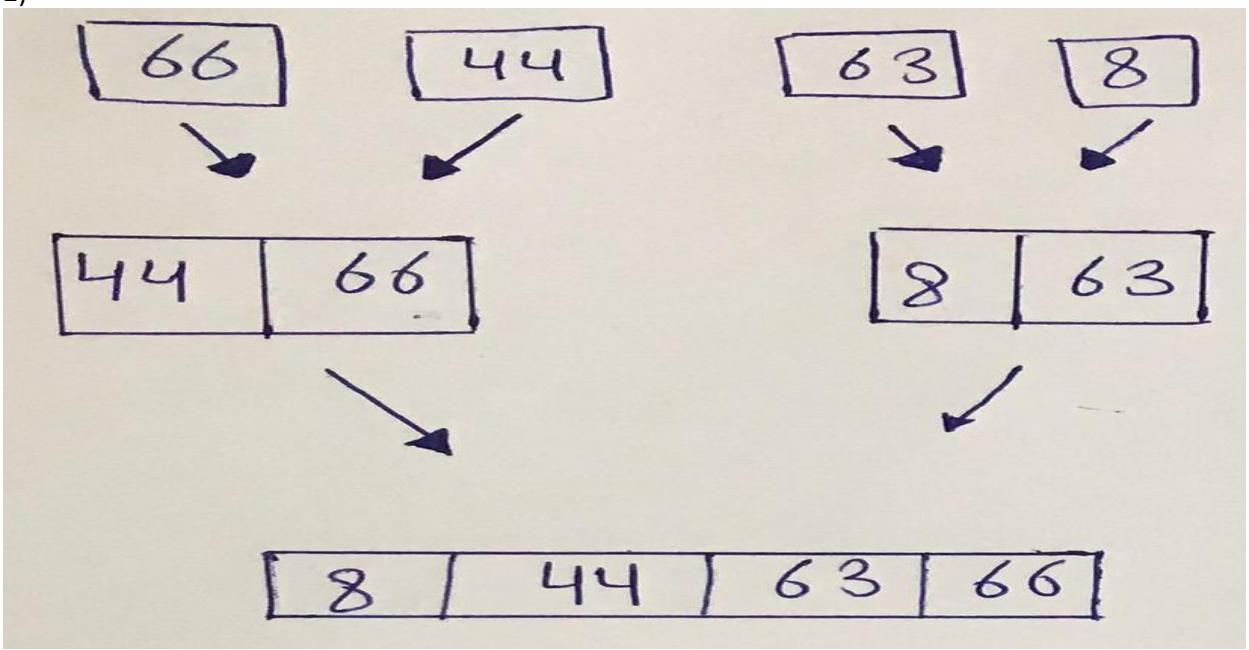
This operation is to do the sort of the elements in the array and merge , let's discuss more about how the sort will work:

1)



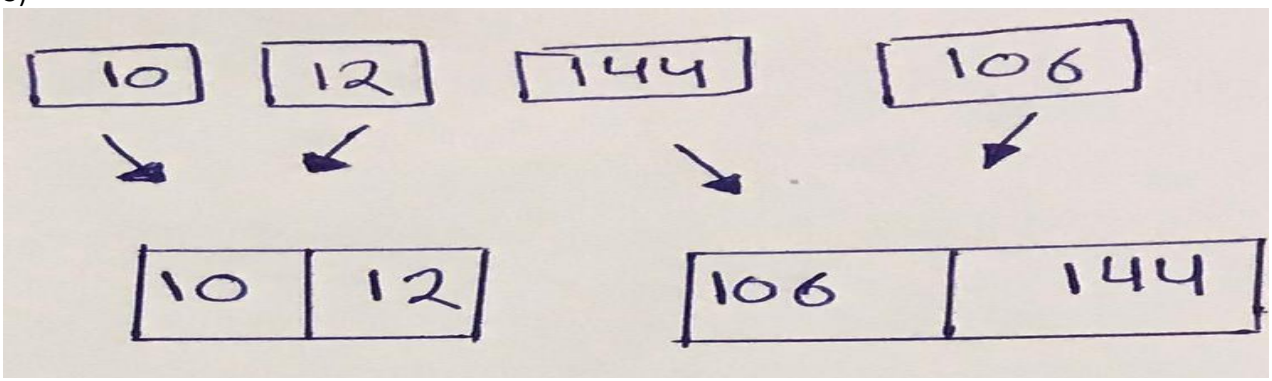
We will take first (66,44) and sort it -> in the left array there is (66) and right array (44) and notice that 44 < 66 so the we will sort it in the new array with the new sort [44,66] as shown in the figure, also take the (63,8) and notice that 8<63 so the sort will become [8,63].

2)

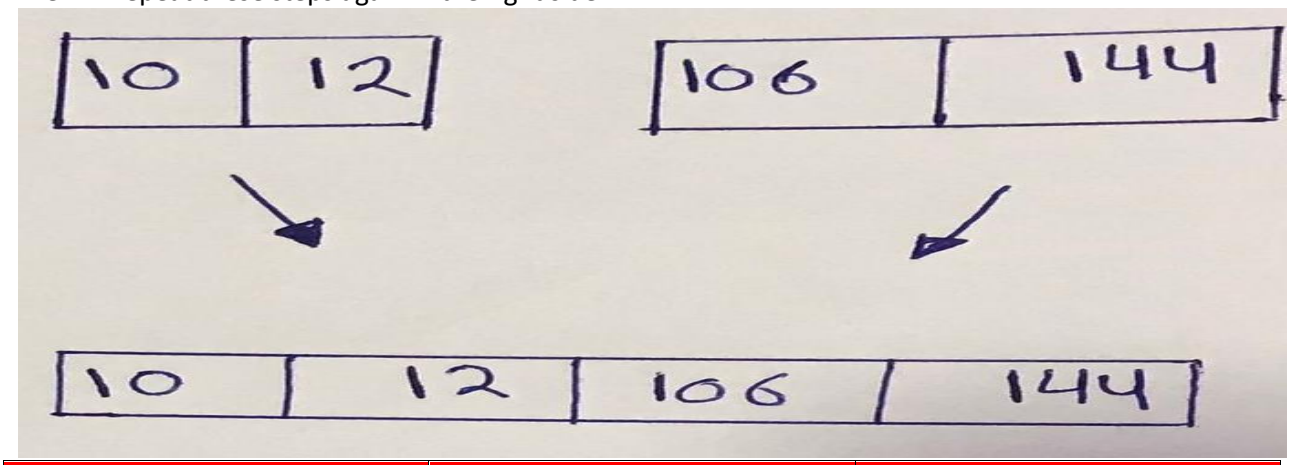


In this point we will sort two arrays and each array has two elements and in this point, we will sort only the unpicked elements. First we will start with (44) from the left and (8) from the right and sort it, notice that 8<44 and each of two numbers are unpicked so we will put (8) at the first index of the array sorted as shown in the figure and number (8) will consider as picked number. Also we will take (44) from the left because it still unpicked number and compare it with (63) from the right and notice -> 44 <63 so 44 will put in the second index of the sorted array and consider it as a picked number. Also compare the 66 from the left and 63 from the right and compare it -> 63<66 so we will put 63 in the third index and 66 in the fourth index of the array and the final result will be as shown in the figure above.

3)

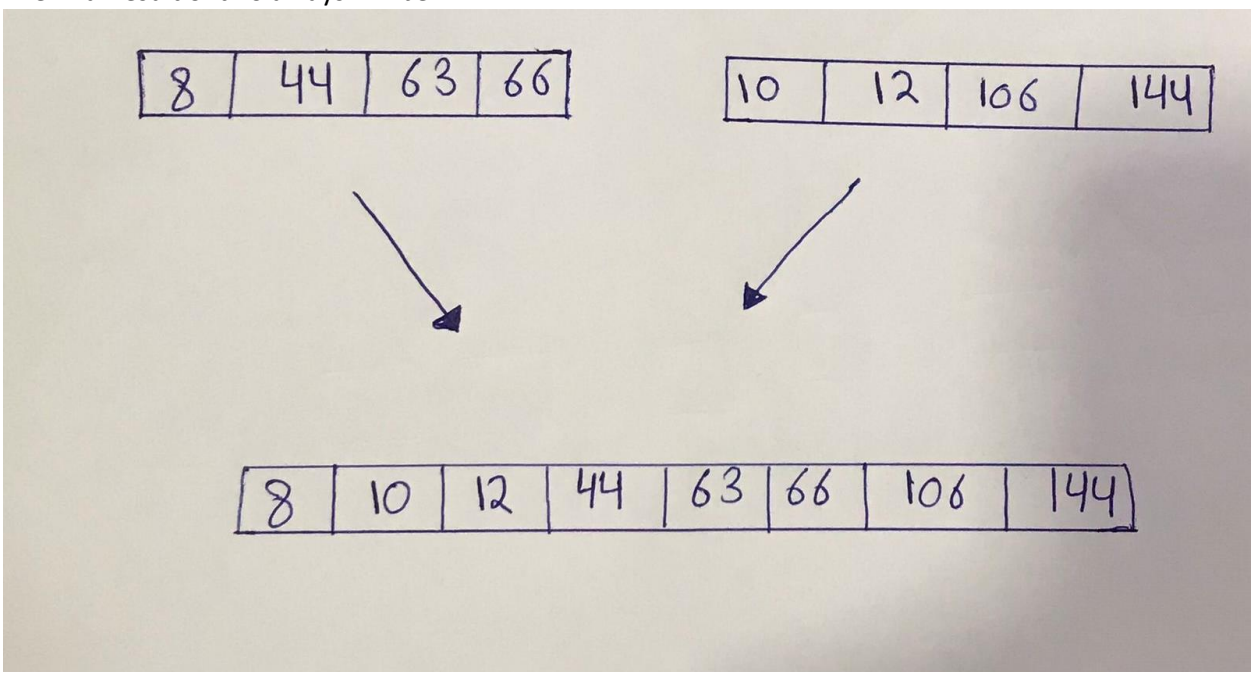


We will repeat these steps again in the right side.

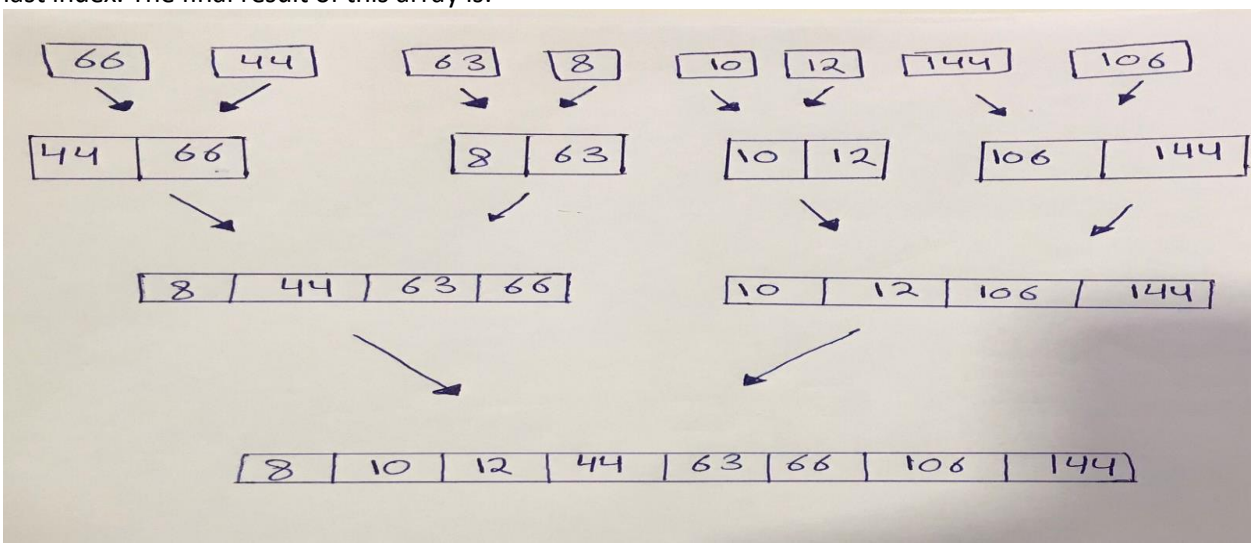


Compare	visited	Uninvited
10 -> 106	10	106
12->106	12	106
	106	
	144	

4)
The final result of two arrays will be:



So again , first take (8) from the left and (10) from the right and compare, noticed that 8 <10 so number 8 will put in the first index of the sorted array and consider the number 8 as visited. Take (44) left and (10) right and compare-> 10 < 44 so number 10 will be in the second index of the sorted array and consider it as a visited number, (44) left with (12) right -> 12 < 44 so number 12 will be in the third index of the sorted array and consider it as a visited number, (44) left with (106) right -> 44 <106 so 44 will be in the fourth index of the sorted array and consider it as a visited number,(63) left and (106) right -> 63 <106 so 63 will be the fifth index of the sorted array and consider it as a visited, (66) left and (106) right -> 66 <106 so 63 will be the sixth index of the sorted array and consider it as a visited. Noticed that the unpicked number are (106 and 144) and the two numbers in the right array, and the left array has no elements so we will put them as is in the sorted array so 106 will be in the seventh index and 144 in the last index. The final result of this array is:



So in general, the main operation are:
-Divide the array into sub problems and deal with each sub problem by using recursion. In my program -> recursive(Array).

-Merge the sub arrays into one sorted array . in my program-> merge_sort (left_array,right_array,Array).

Also the merge_sort function will take input -> three arrays which are left array , right array and Array and the output is one array.And recursive function will also take input -> Array and give output : if the size of the array at first is less than 2 , the output is one array but if the size of the array is big than 2, it will give at least 3 arrays (left, right,Array) .

ENCAPSULATION AND INFORMATION HIDING

To discuss more about the encapsulation in programming , let's know what is the encapsulation:
In programming-> it is the method that aims to protect and hide the information by mixing the element to make a new entity. In object oriented-> it is an attribute in of object designed also aims to protect the data in the object by making the data of the object hidden in the object so it will be hard to access it.

- ✧ Benefits :

- ❖ Protects the object from any suspect access.
- ❖ It also decrease the human errors.
- ❖ Easily in maintenance the application .
- ❖ Gives the users to understand the application easily.

(What Is Encapsulation ?, 2022)

Also from the research, there is also something called (information hiding) -> hide the information such as : programmable code internally which designed to be private.Benefit-> it gives the programmers the flexibility by modify the programme and also, you can replace the source code inside modules to give you to access it in the future and more easier to reuse.

(What is Information Hiding? - Definition from Techopedia, 2022)

ADT AND OBJECT ORIENTED

We all know what is the ADTs in data structure--> Brief description about a specific problem in data structure or algorithm about the main operations in the algorithm and how it's work for many benefits for the developer. And let's discuss more about object oriented in programming generally which is a concept that depends on the objects and classes, and it used for many reasons but the main reason is to create a simple software and can reused the code.

From the research, I found many important information about abstract data type and object oriented, for example I found that the oldest paradigms are relied on von Neumann architecture and it is the imperative paradigm, a lot of programming paradigms developed by using the imperative.After that procedural paradigms came and it is an best version of imperative because it can solve a complex problems without facing any problems, then came ADT which is considered as additional extension of imperative paradigms. So object oriented (which contains of classes) pardigmes extendes ADT which is the better of imperative programming languages, the data and operations are encapsulated in the objects. By solving all the issues of latest developed programming paradigms -> the object oriented paradigms was developed. But the first imperative language is the machine language , after that procedural came, then the ADT which is considered as the development of procedural method and also ADT is special because it has the main feature -> data hiding and encapsulation.

(What is imperative ADT? Is abstract data type in data structure similar to abstraction in object orientation?, 2022)

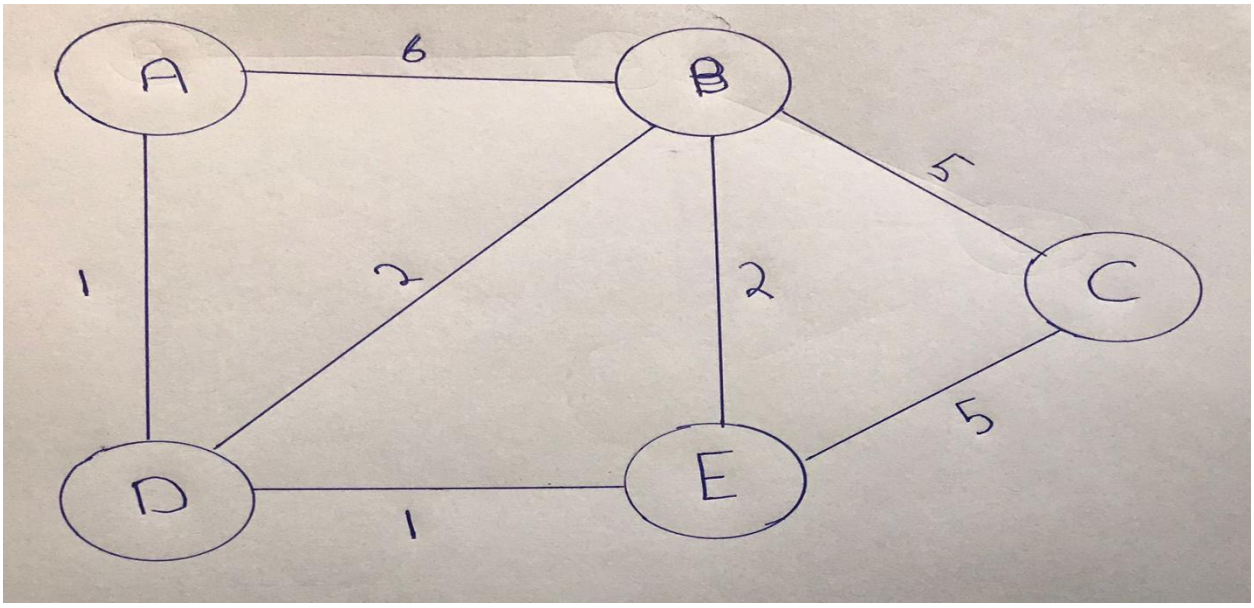
Extra:
and also from the research I do, I found that the ADT is basic of object oriented so it's a correct view because from the description about (OOP) above, The design of object oriented is depending on -> the applications need to be created around classes and objects. For example (class) -> is abstract of its members (attributes and methods in it). From all these information above, I strongly agree that the imperative ADT is the basis of object oriented.

(Data Structures and Algorithms: Objects and ADTs, 2022)

TASK TWO:

ANALYSE OPERATIONS OF TWO SORTING ALGORITHMS

First let's talk about the **Dijkstra's Algorithm** --> which is an algorithm that created by a Dutch scientist, it created for a specific aim which is ti find the shortest path between two nodes in a graph by providing the starter node to all the nodes in the graph.To clear it more, let's take an example of the algorithms and how it works:



If we have this graph with the the possible paths as shown in the figure and we need to find the shortest path by using the algorithms.First , we need to identify the starter vertex and let's consider (A --> starter vertex).

Vertex	Shortest path	Previous Vertex
A	0	

Unvisited =[A,B,C,D,E] Visited=[]

Once it run , the algorithm will make the most important information that we will need to know as shown in the table, in the table we have three columns, in the first column it will contains all the vertexes in the graph , in the middle it will contains the shortest path between the the starter vertex (A) to another vertexes in the graph, in the third column it will contains the short sequences from A to another.So the first thing, take the starter vertex (A) and put it in the table with the shortest path is zero and doesn't have any previous vertex.and show all the neighboring vertices

Vertex	Shortest path	Previous Vertex
A	0	
B		
D		

-Show all the neighboring vertices to A which are : B and D as shown in the graph above and put it in the

Vertex	Shortest path	Previous Vertex
A	0	
B	3	D
D		

vertex column.

- Find the shortest path between A -> B and we have two paths: 1) A->B (path =6) 2) A->D->B (path=3) So we will choose the second option because it's shorter than the one option, and put the shortest path in the middle column and we chose the second option so the previous vertex will be D and put it in the third column.

Vertex	Shortest path	Previous Vertex
A	0	
B	3	D
D	1	A

Unvisited =[B,C,D,E] Visited=[A]

- We will make the same thing between A and D and find the shortest path which is from A-> D (with path =1) and the previous vertex is (A) and put these information in the table as shown.After we finished from the neighboring vertices of A , we will put the A from the Unvisited group to visited group.

Vertex	Shortest path	Previous Vertex
A	0	
B	3	D
D	1	A

- Choose the shortest path from the table (Except the visited group such as A) which is vertex (D) and find all neighboring vertices.

Vertex	Shortest path	Previous Vertex
A	0	
B	3	D
D	1	A
E		

-The neighboring vertices of D are -> A,B,E, but we will not count the visited group, so we will not count the A-->(B and E).

Vertex	Shortest path	Previous Vertex
A	0	
B	3	D
D	1	A
E		

- Find the shortest path between the D and B which is the path between D to B (with path = 2) but in the table, the shortest path is equal to 1 so we will not take the path between D to B because there is another path that is shortest (path =1).

Vertex	Shortest path	Previous Vertex
A	0	
B	3	D
D	1	A
E	2	D

Unvisited =[B,C,E] Visited=[A,D]

- Also find the shortest path between the D and E which is the path between D and E (with path = 1) but we will not put 1 in the shortest path and it will become (2) because (D =1) and the path between (D and E =1) so the final path will become = 1+1= 2 and the previous vertex of E is D . Put the D in the visited group.

Vertex	Shortest path	Previous Vertex
A	0	
B	3	D
D	1	A
E	2	D

-Again , we will take the shortest path except the visited group, so we will take E and find all the neighboring vertices.

Vertex	Shortest path	Previous Vertex
A	0	
B	3	D
D	1	A
E	2	D
C		

- The neighboring vertices of E are : C,B,D , and again we will not count D because it's in the visited group.

Vertex	Shortest path	Previous Vertex
A	0	
B	3	D
D	1	A
E	2	D
C		

- Find the shortest path between E and B which is between E-> B with path =2+2=4, but in the table , the previous shortest path (3) is shorter than (4) so we will keep the shortest path in B as before.

Vertex	Shortest path	Previous Vertex
A	0	
B	3	D
D	1	A
E	2	D
C	7	E

Unvisited =[B,C] Visited=[A,D,E]

-Find the shortest path between E and C which is between E->C (with path =5+2=7)--> newpath = the path between E to C + shortest path in E , and the previous path is E. Put the E in visited group.

Vertex	Shortest path	Previous Vertex
A	0	
B	3	D
D	1	A
E	2	D
C	7	E

- Select the shortest path from the unvisited group which is B and find the neighboring vertices except the visited group.

Vertex	Shortest path	Previous Vertex
A	0	
B	3	D
D	1	A
E	2	D
C	7	E

-The neighboring vertices are --> only c because the others are in visited group.

Vertex	Shortest path	Previous Vertex
A	0	
B	3	D
D	1	A
E	2	D
C	7	E

Unvisited =[C] Visited=[A,D,E,B]

-Find the shortest path between the B and C which is (5+3=8) but the previous path in the table is shorter, so we will keep it.Put the B in the visited group.

Vertex	Shortest path	Previous Vertex
A	0	
B	3	D
D	1	A
E	2	D
C	7	E

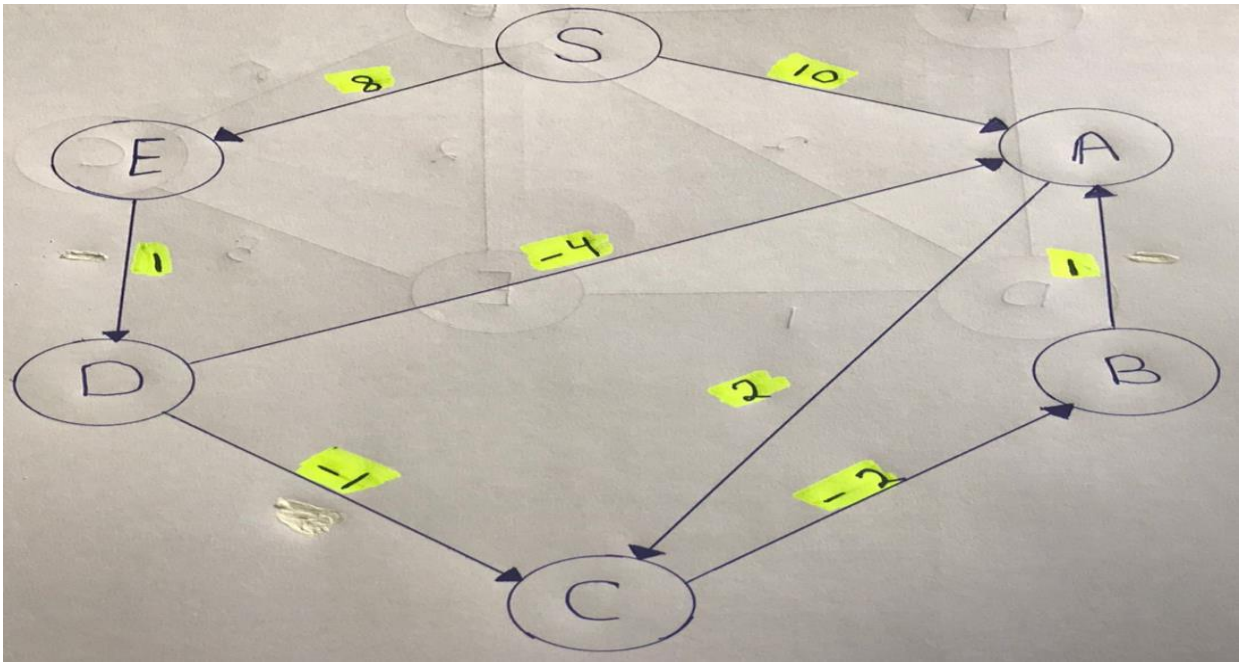
Unvisited =[] Visited=[A,D,E,B,C]

-And finally because the C is the final vertex, we will put it in the visited group without any changes in the table.And that it how the Dijkstra's Algorithm works.

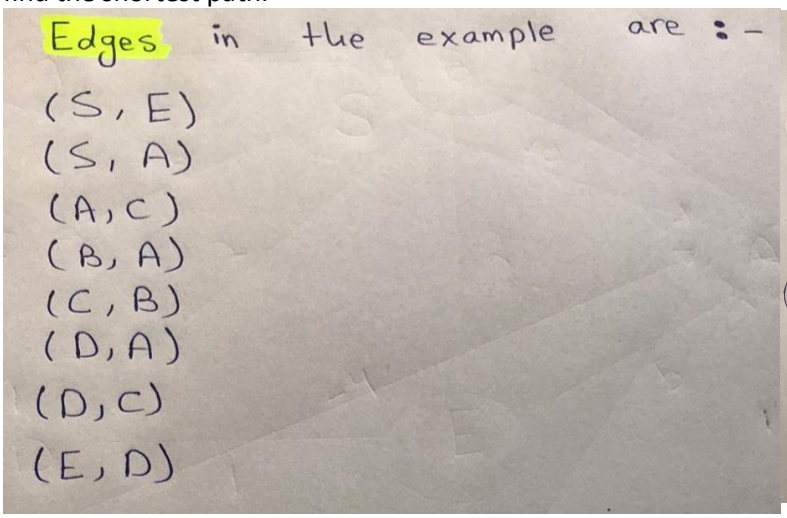
Note This algorithm can't deal with the negative numbers and also if it deals with these numbers, we can't confirm it as a correct algorithm to find the shortest path when we deal with the negative numbers. In this case we want algorithm that deals with the negative number and give us the correct shortest path, so we need to talk about **Bellman ford algorithms**.

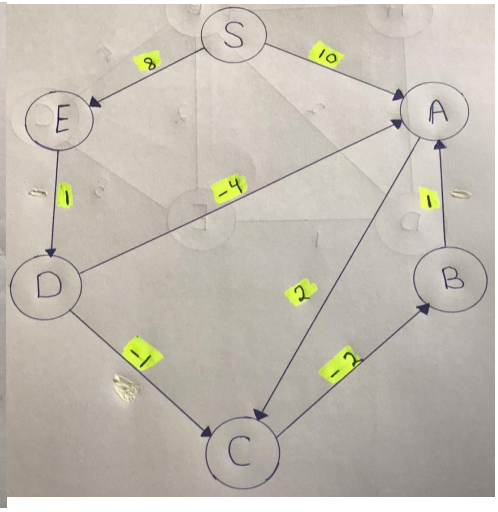
Bellman ford is same as functionality as Dijkstra's algorithm because the two give us the shortest path by finding the shortest path between one vertex to all vertices in the graph but the difference between the two mentioned in the note which is "negative numbers".

In conclusion of Bellman ford : if we have a graph and it contains -> 6 vertices, in this algorithm, we will relax in 5 edges (number of vertices - 1), so we will repeat the the steps 5 times.

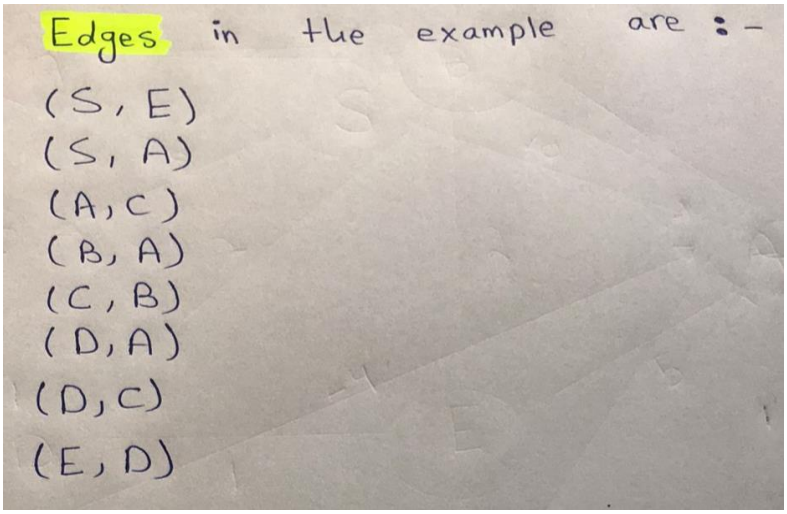


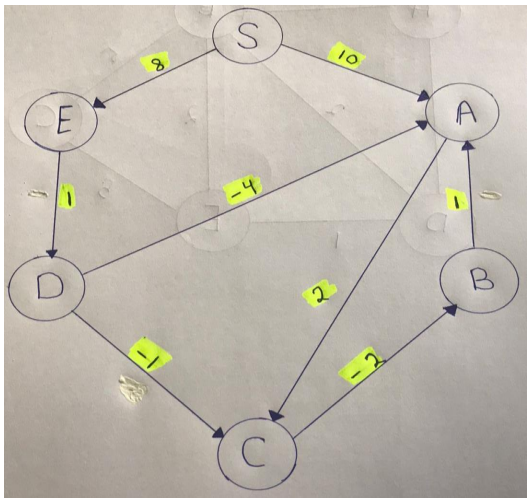
If we have this graph and we need to find the shortest path, we can't use the previous algorithm because there are a negative weighted(negative numbers) so we will use the Bellman ford algorithm to find the shortest path.





-First step considered the start vertex is (S)--> is we need to create a list that contains all the edges in the graph (you can change the order of the edges for example you can put the edge (S,A) at the first and so on).

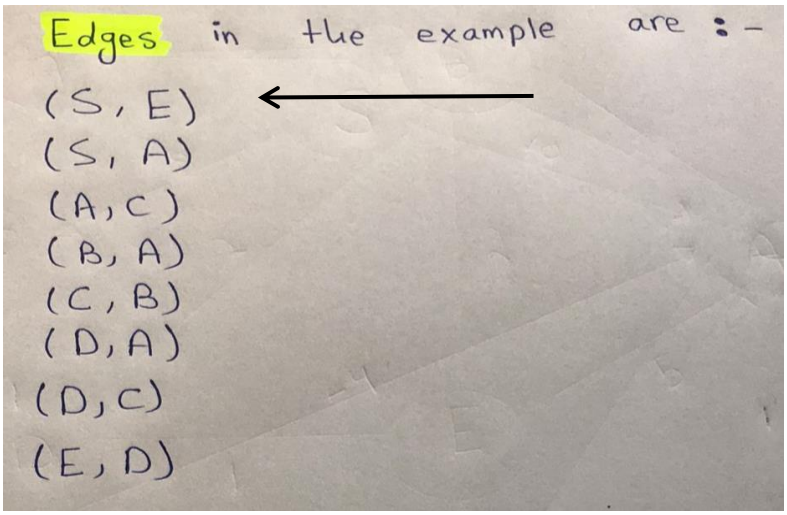


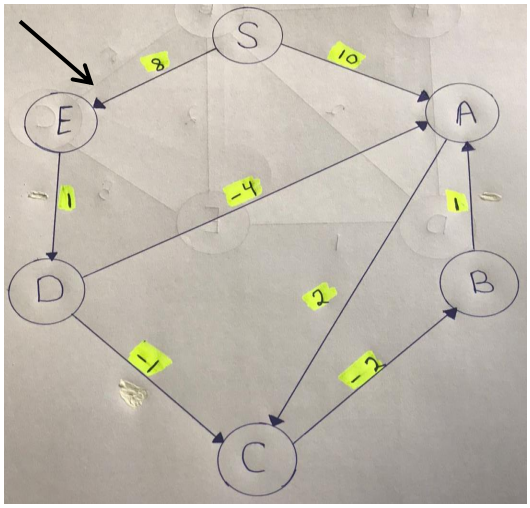


Vertex	Distance
S	0
A	∞
B	∞
C	∞
D	∞
E	∞

-As shown in the table, the list is made and it contains all the vertices and distance of each vertex to find the shortest path, set the distance at (S) is equal to zero because it's the start vertex and the other vertices are equal to infinite.

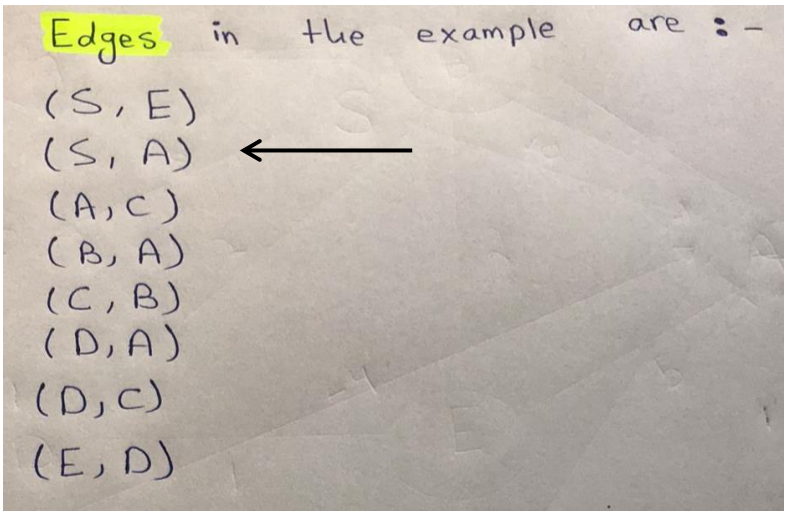
-Second step is finding the relax as discussed before, in this example we have 6 vertices -> so the we will relax all the edges 5 times.

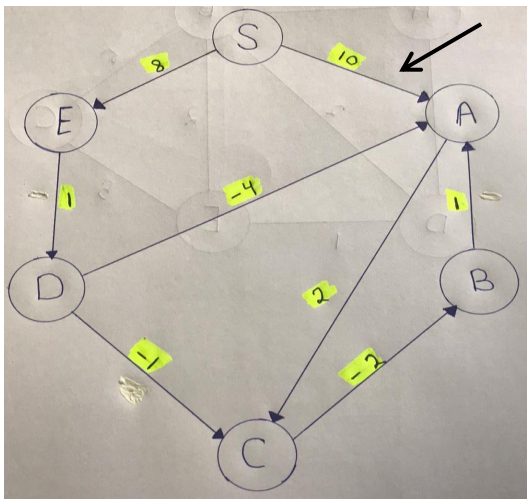




Vertex	Distance
S	0
A	∞
B	∞
C	∞
D	∞
E	8

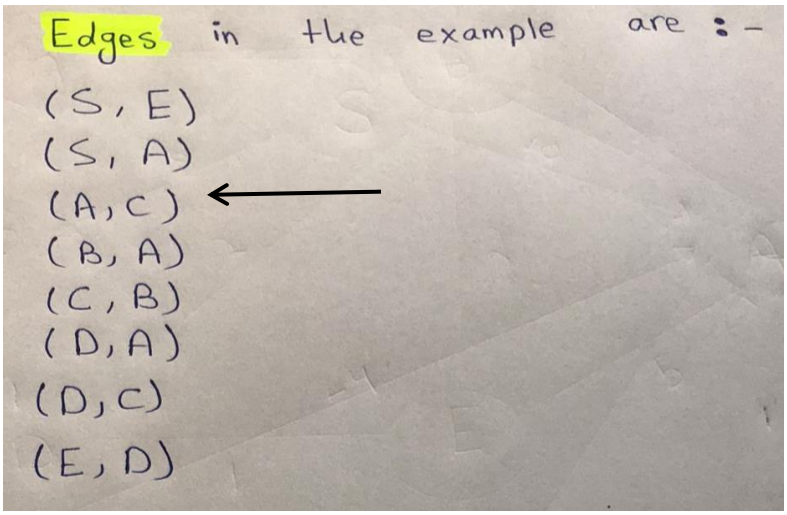
- **First iteration** with the first edge (S,E) as shown in the figure and find the shortest distance. From the table, the distance of vertex(S) is equal to zero and if we want to move to vertex (E) the distance will be (0+8=8), and compare (8) with the distance at vertex E which is infinite, if the new distance of vertex is smaller than the distance of the same vertex in the table, we will update the table and set the the new distance in the table. So from the first iteration -> new distance of vertex E= (8) and the distance of E from the table is infinite --> 8 < ∞ ,so the 8 will put as a new distance of vertex E.

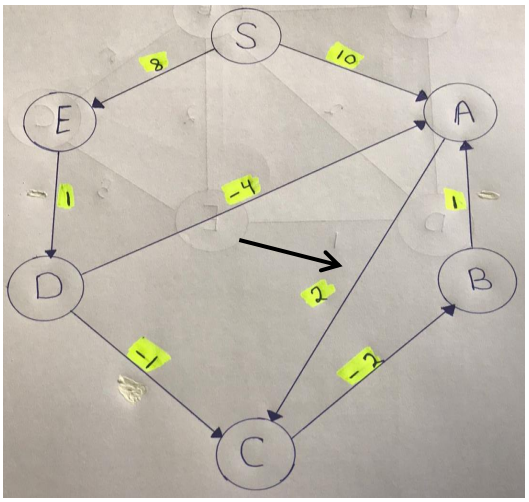




Vertex	Distance
S	0
A	10
B	∞
C	∞
D	∞
E	8

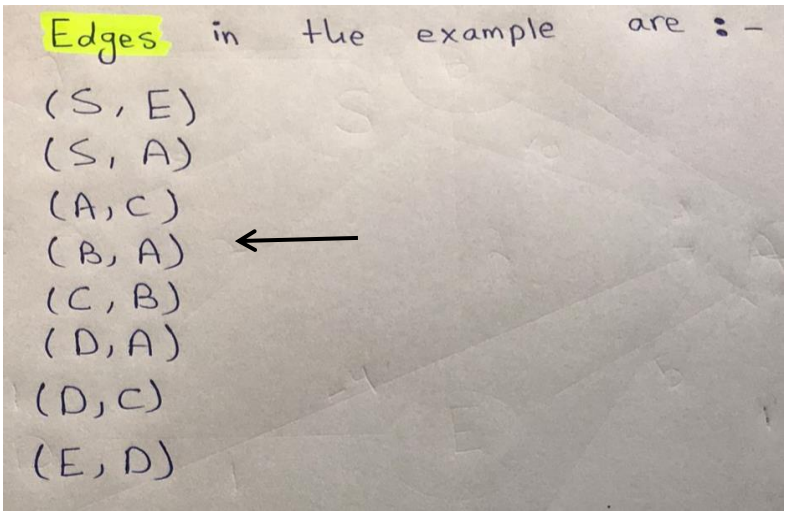
- the second edge is (S,A), the new distance of vertex A =(0+10=10) and the distance of vertex A from the table = ∞ ,and 10 <∞ , so we will put (10) as a new distance value in the table.

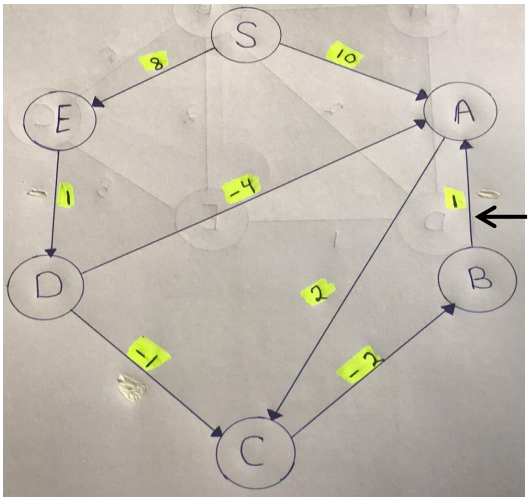




Vertex	Distance
S	0
A	10
B	∞
C	12
D	∞
E	8

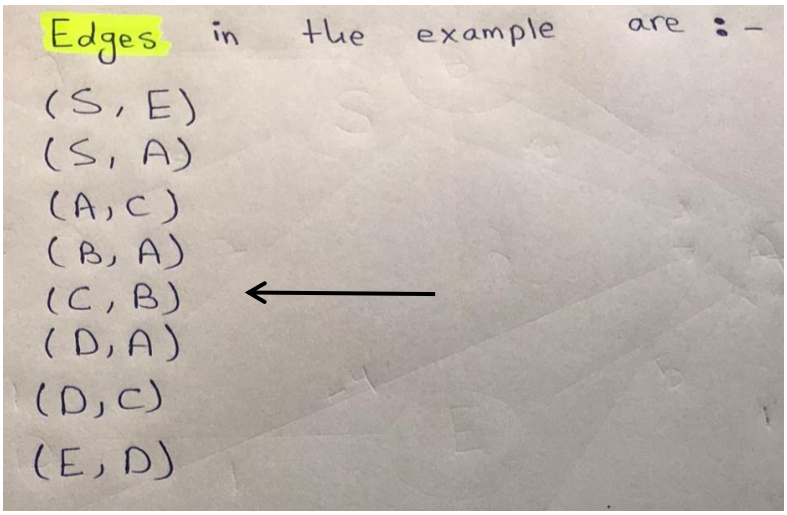
- Third edge is (A,C), the new distance of vertex C =(10(value of vertex A)+2(edge)=12) and the distance of vertex C from the table = ∞ ,and 12 <∞ , so we will put (12) as a new distance value in the table.

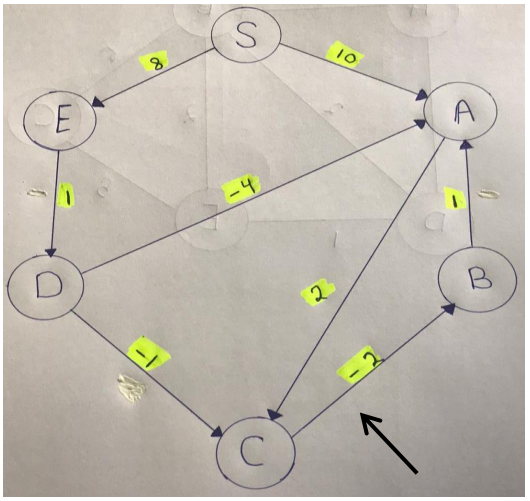




Vertex	Distance
S	0
A	10
B	∞
C	12
D	∞
E	8

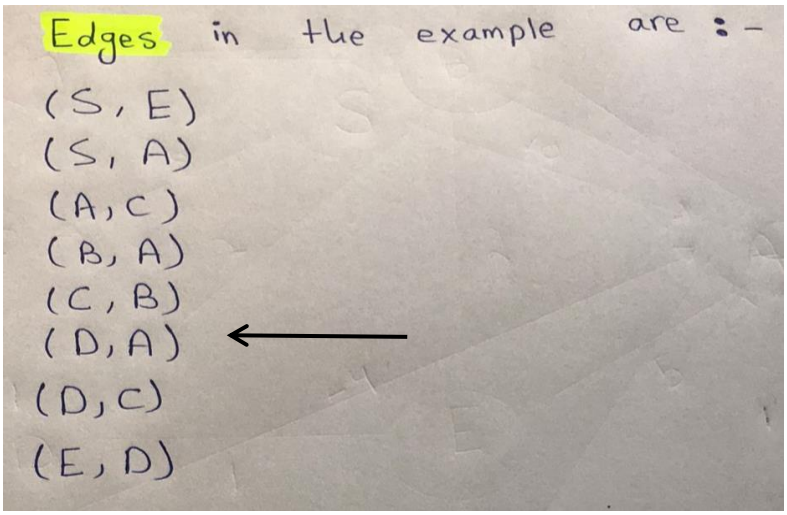
-Fourth edge (B,A) the new distance of vertex A=(∞(value of vertex B)+1(edge)=∞) and the distance of vertex A from the table = 10 ,and ∞>10 , so we will not put (∞) as a new distance value in the table because the new distance are not smaller than the old distance (from the table) and keep the old distance in the table.

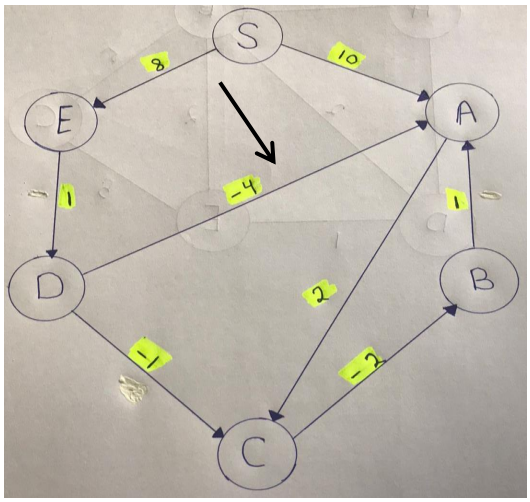




Vertex	Distance
S	0
A	10
B	10
C	12
D	∞
E	8

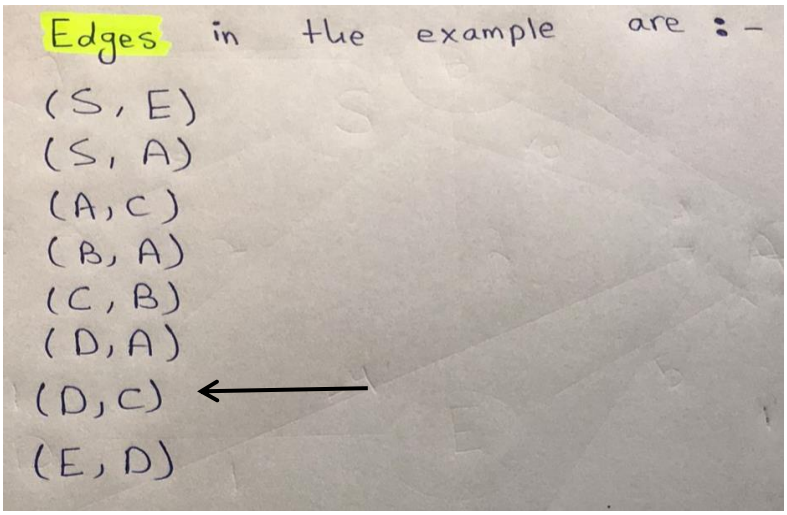
-Fifth edge (C,B), the new distance of vertex B=(12(value of vertex C)-2(edge)=10) and the distance of vertex B from the table = ∞ ,and 10 <∞ , so we will put (10) as a new distance value in the table.

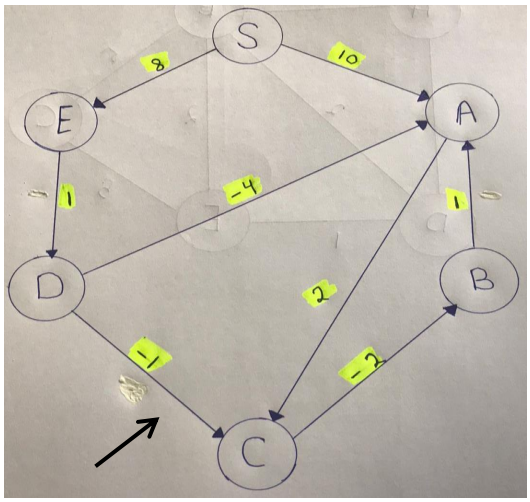




Vertex	Distance
S	0
A	10
B	10
C	12
D	∞
E	8

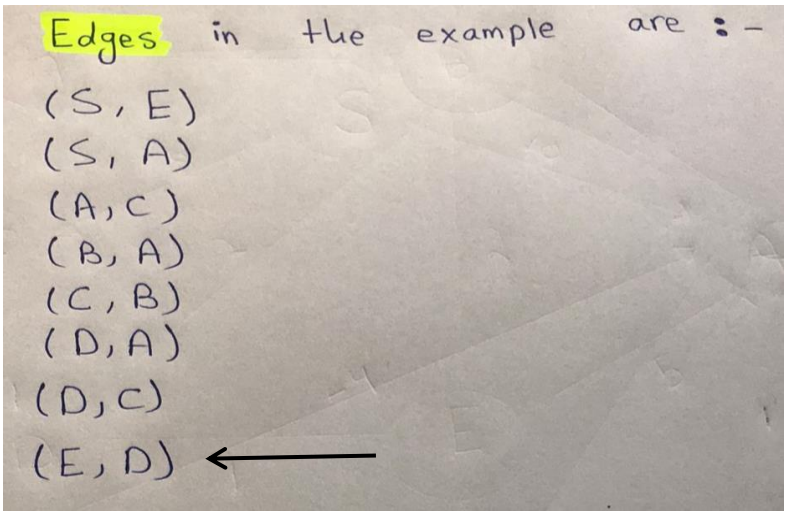
-Sixth edge(D,A),the new distance of vertex A=(∞(value of vertex D)-4(edge)=∞) and the distance of vertex A from the table = 10 ,and ∞>10 , so we will not put (∞) as a new distance value in the table because the new distance are not smaller than the old distance (from the table) and keep the old distance in the table.

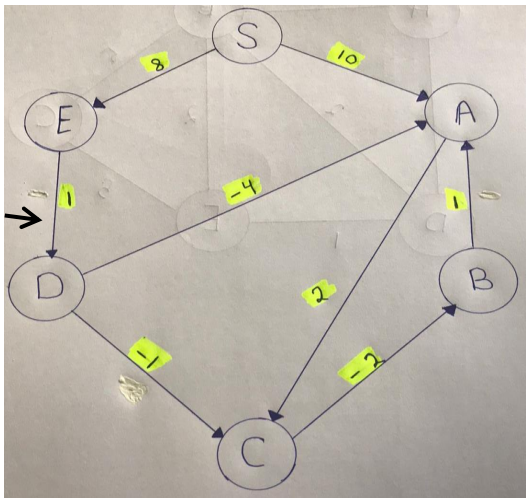




Vertex	Distance
S	0
A	10
B	10
C	12
D	∞
E	8

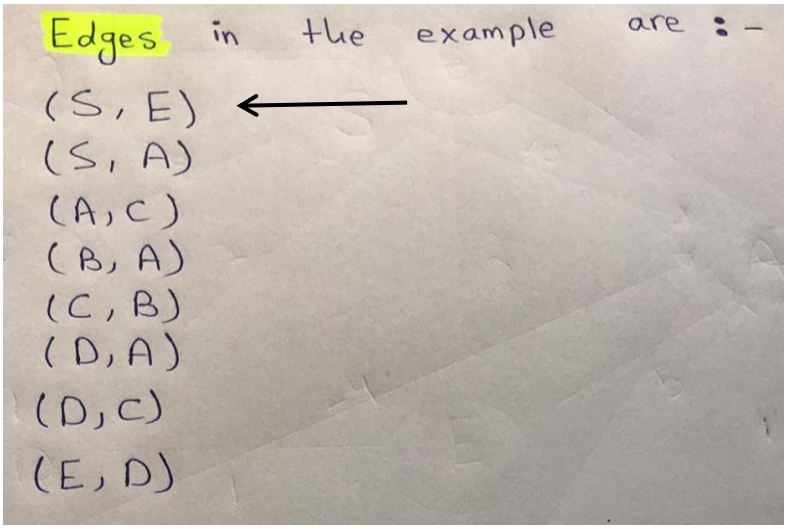
-Seventh edge (D,C),the new distance of vertex C=(∞(value of vertex D)-1(edge)=∞) and the distance of vertex C from the table = 12 ,and ∞>12 , so we will not put (∞) as a new distance value in the table because the new distance are not smaller than the old distance (from the table) and keep the old distance in the table.

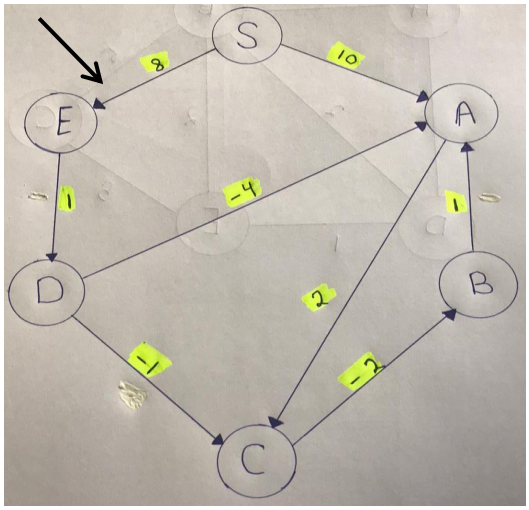




Vertex	Distance
S	0
A	10
B	10
C	12
D	9
E	8

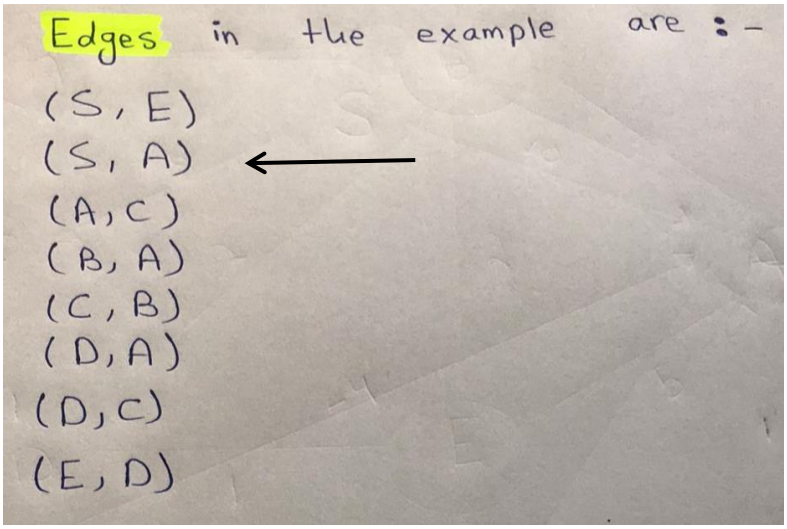
- Eight edge, the new distance of vertex D =(8(value of vertex E)+1(edge)=9) and the distance of vertex D from the table = ∞ ,and 9 <∞ , so we will put (9) as a new distance value in the table.

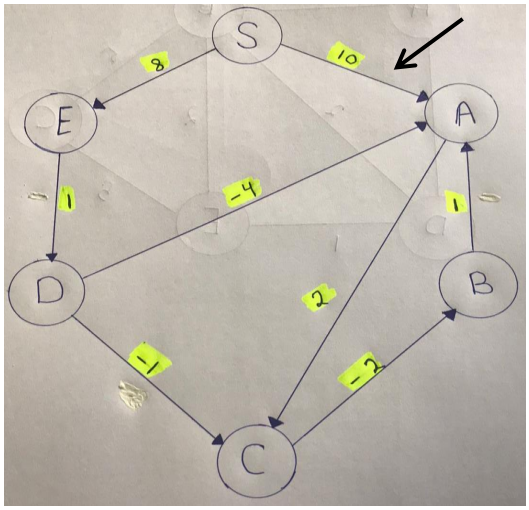




Vertex	Distance
S	0
A	10
B	10
C	12
D	9
E	8

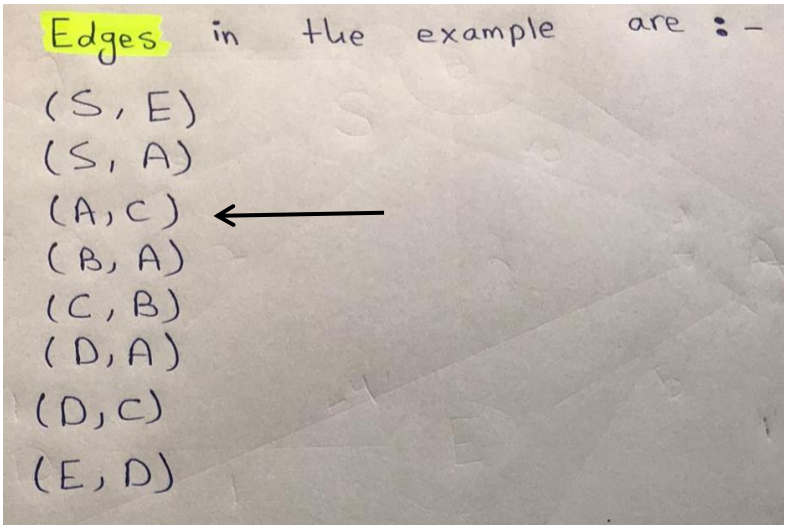
- **Second iteration,** in second iteration we will repeat the steps before to check if we reached the shortest path until reach the iteration number five(if needed). In first edge (S,E) ,the new distance of vertex E (0+8 =8) and the old distance from the table = 8 , and 8=8 so the value will not change.

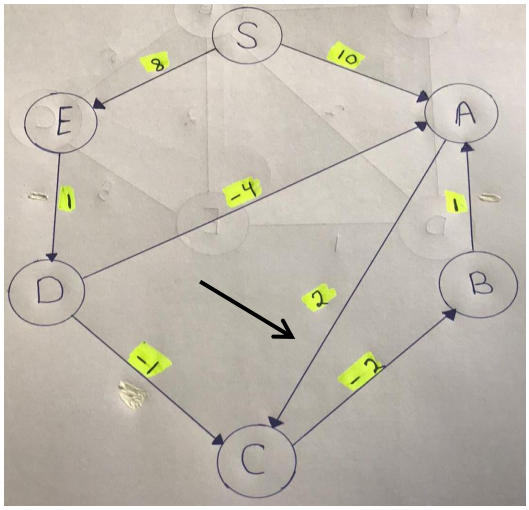




Vertex	Distance
S	0
A	10
B	10
C	12
D	9
E	8

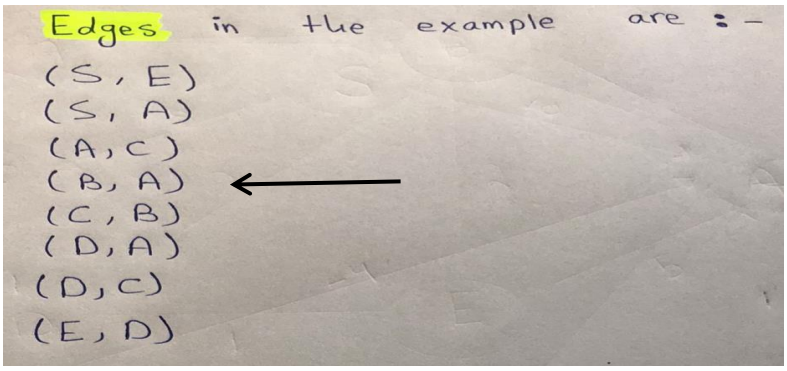
-In second edge (S,A) ,the new distance of vertex A (0+10 =10) and the old distance from the table = 10 , and 10=10 so the value will not change.

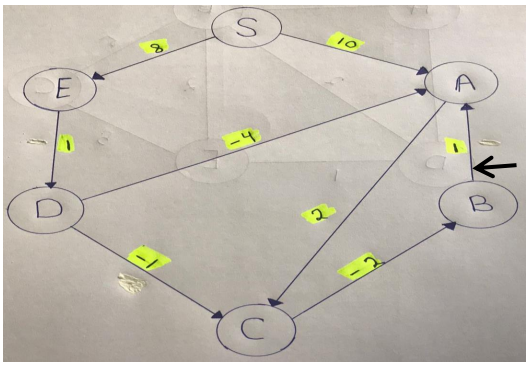




Vertex	Distance
S	0
A	10
B	10
C	12
D	9
E	8

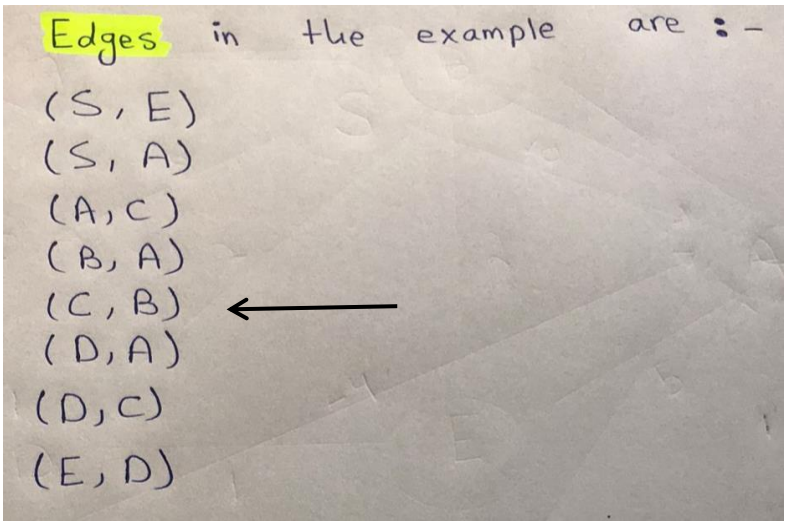
-In third edge (A,C) ,the new distance of vertex C (10+2 =12) and the old distance from the table = 12 , and 12=12 so the value will not change.

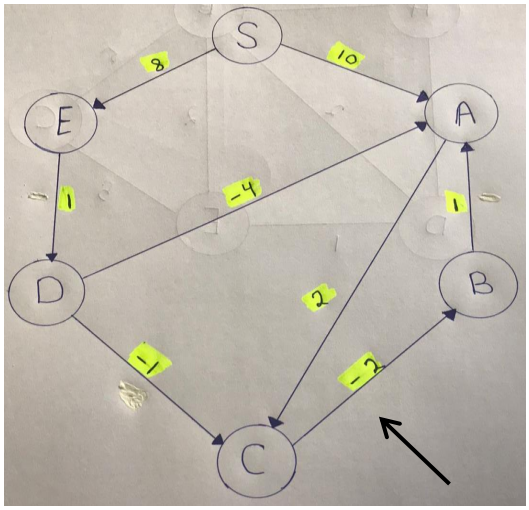




Vertex	Distance
S	0
A	10
B	10
C	12
D	9
E	8

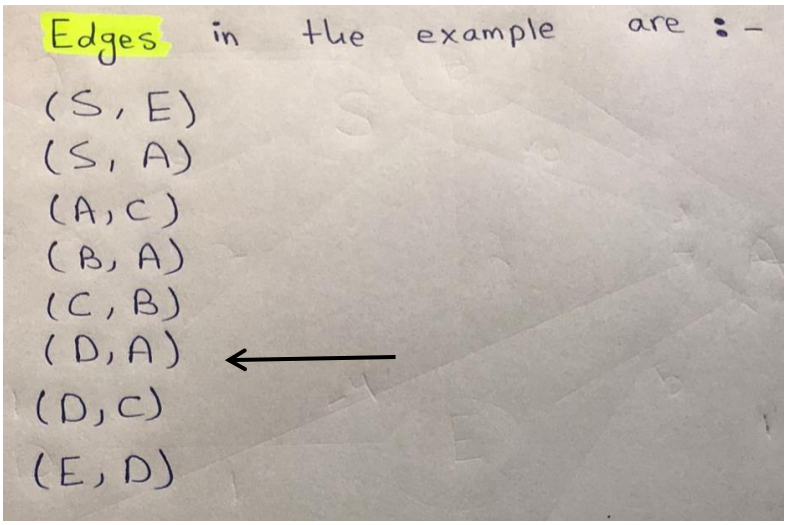
-In fourth edge (B,A) ,the new distance of vertex A (10+1 =11) and the old distance from the table = 10 , and 10 < 12 so the value will not change.

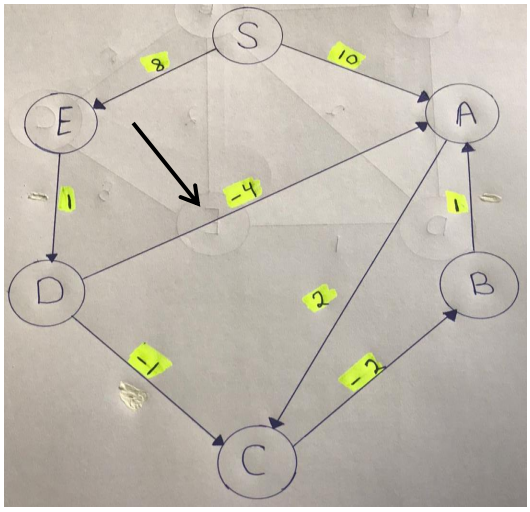




Vertex	Distance
S	0
A	10
B	9
C	12
D	9
E	8

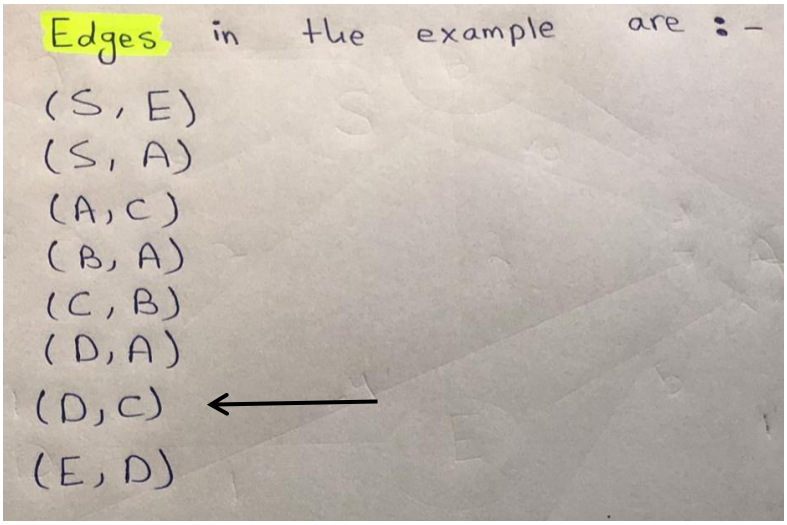
-In fifth edge (C,B) ,the new distance of vertex B (12-2 =10) and the old distance from the table = 10 , and 10= 10 so the value of vertex B will not change.

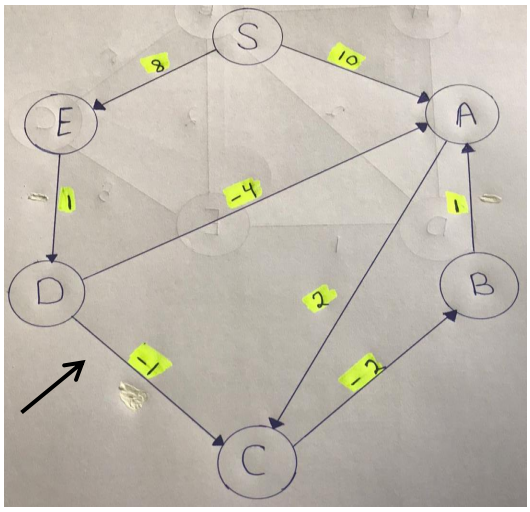




Vertex	Distance
S	0
A	5
B	9
C	12
D	9
E	8

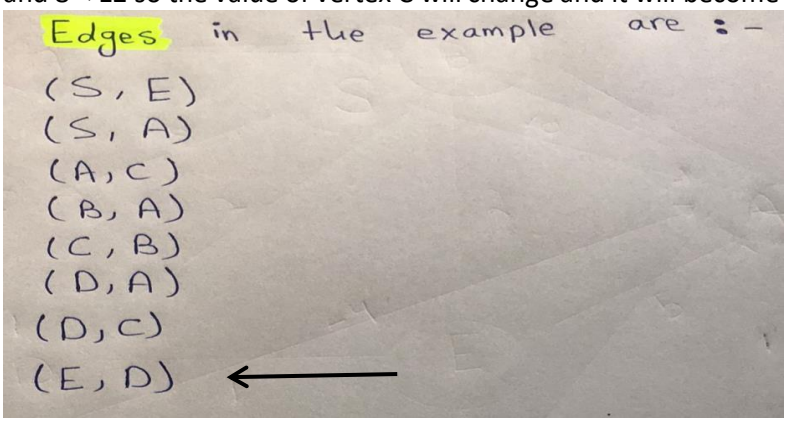
-In sixth edge (D,A) ,the new distance of vertex A (9-4 =5) and the old distance from the table = 10 , and 5 < 10 so the value of vertex B will change and it will become the value of new distance = 5.

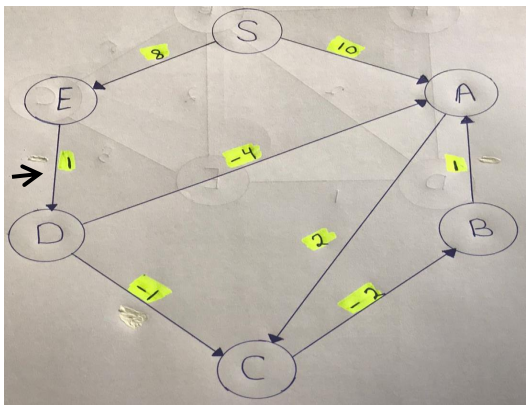




Vertex	Distance
S	0
A	5
B	9
C	8
D	9
E	8

-In seventh edge (D,C) ,the new distance of vertex C (9-1 =8) and the old distance from the table = 12 , and 8 < 12 so the value of vertex C will change and it will become the value of new distance = 8.

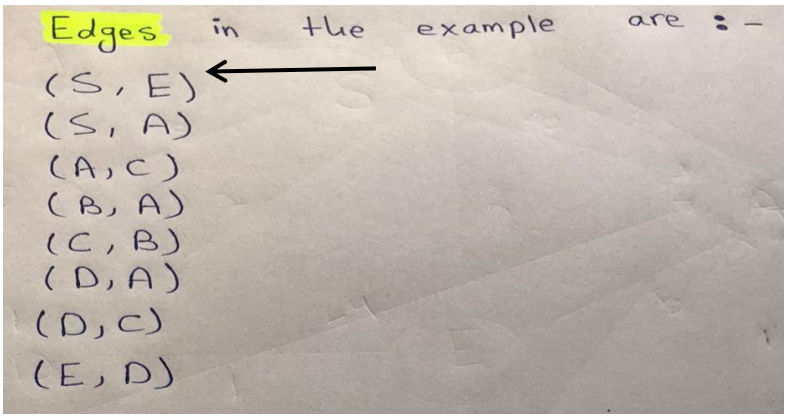


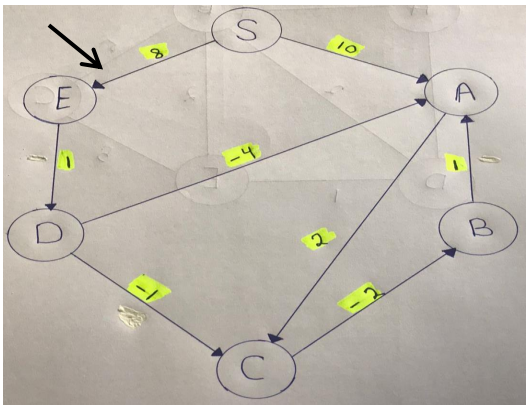


Vertex	Distance
S	0
A	5

B	9
C	8
D	9
E	8

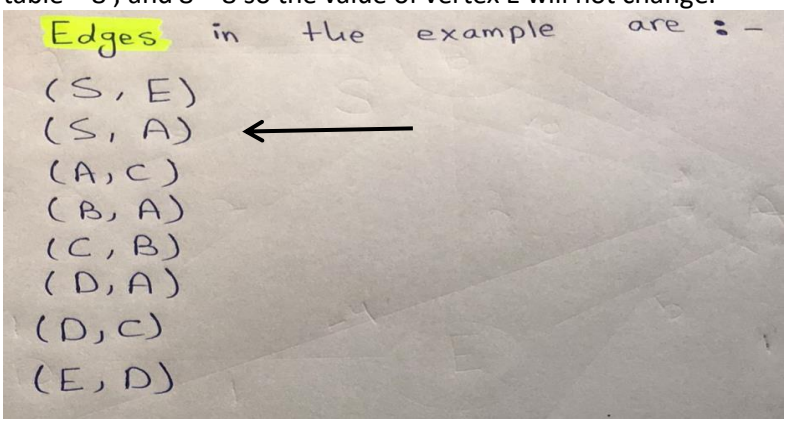
-In eighth edge (E,D) ,the new distance of vertex D (8+1 =9) and the old distance from the table = 9 , and 9 = 9 so the value of vertex D will not change.

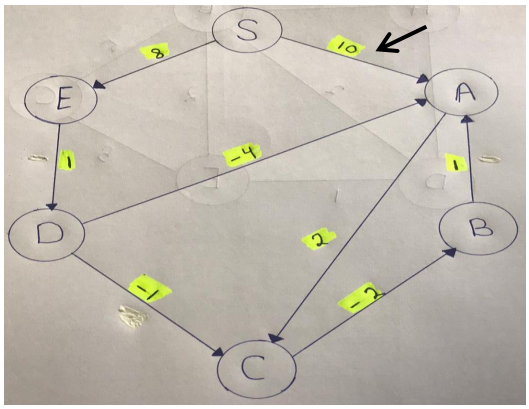




Vertex	Distance
S	0
A	5
B	9
C	8
D	9
E	8

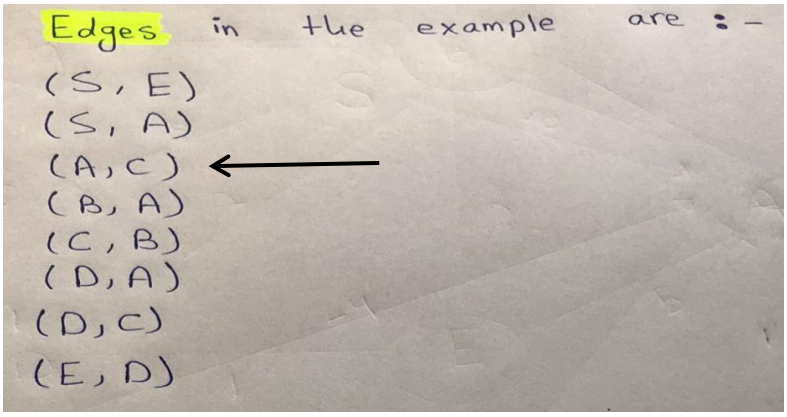
-**Third iteration** in first edge(S,E), ,the new distance of vertex E (0+8 =8) and the old distance from the table = 8 , and 8 = 8 so the value of vertex E will not change.

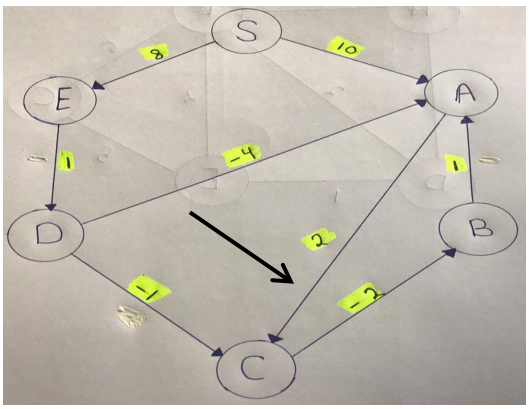




Vertex	Distance
S	0
A	5
B	9
C	8
D	9
E	8

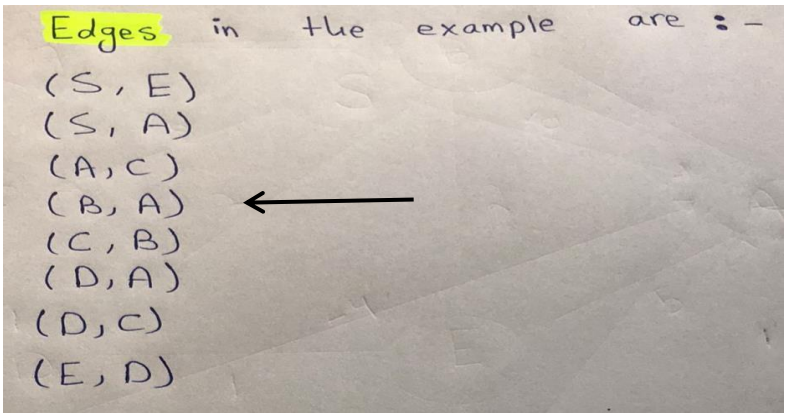
-in second edge (S,A), ,the new distance of vertex A (0+10 =10) and the old distance from the table = 5 , and 5 =< 10 so the value of vertex A will not change.

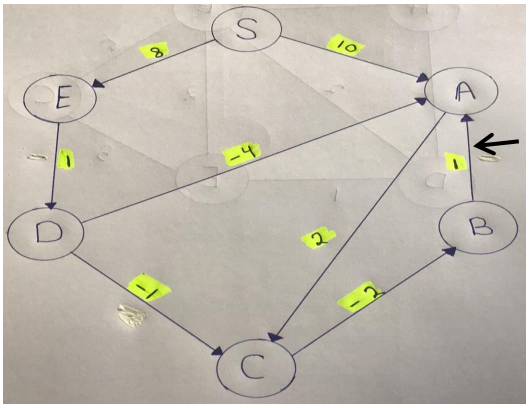




Vertex	Distance
S	0
A	5
B	9
C	7
D	9
E	8

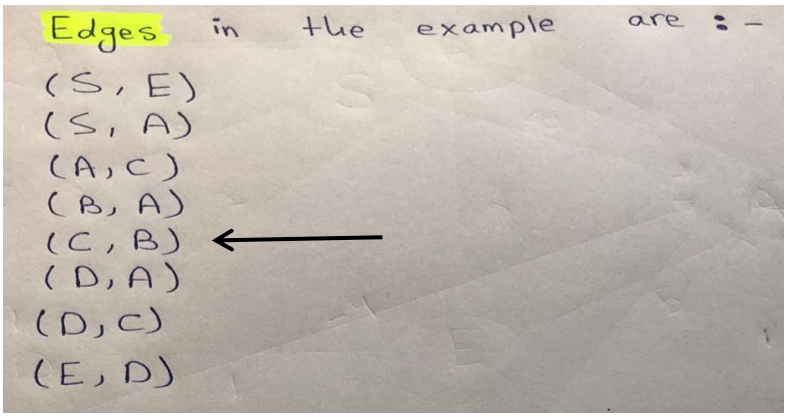
-Third edge (A,C),the new distance of vertex C (5+2 =7) and the old distance from the table = 8 , and 7 < 8 so the value of vertex C will become the same value of the new distance = 7.

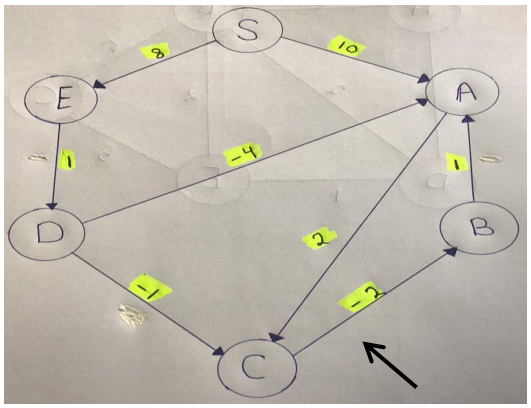




Vertex	Distance
S	0
A	5
B	9
C	7
D	9
E	8

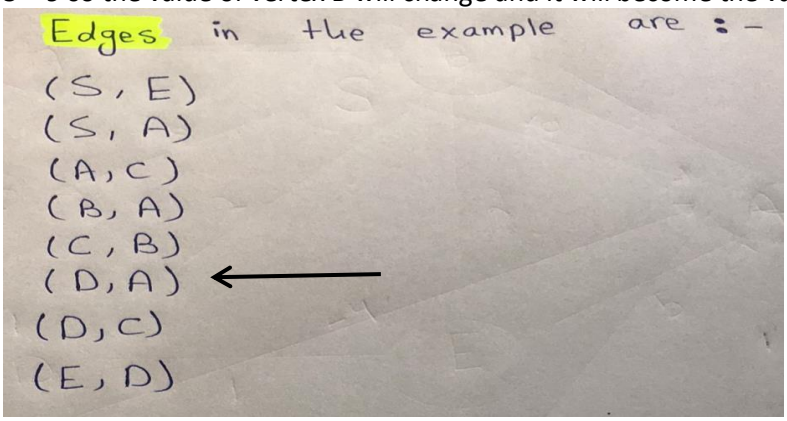
- Fourth edge (B,A), the new distance of vertex A (9+1 =10) and the old distance from the table = 5 , and 5 < 10 so the value of vertex A will not change.

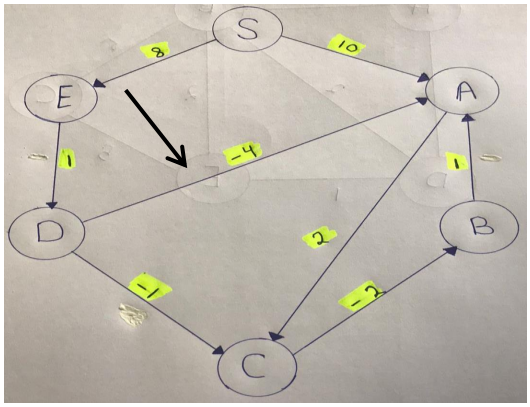




Vertex	Distance
S	0
A	5
B	5
C	7
D	9
E	8

-Fifth edge (C,B), the new distance of vertex B (7-2 =5) and the old distance from the table = 9 , and 5 < 9 so the value of vertex B will change and it will become the value of the new distance = 5.

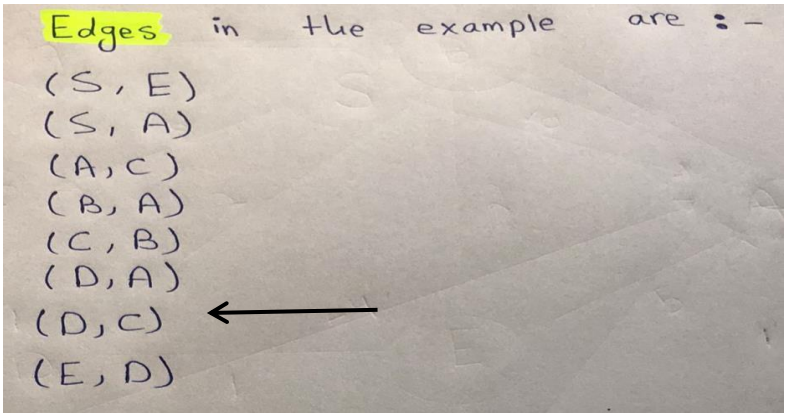


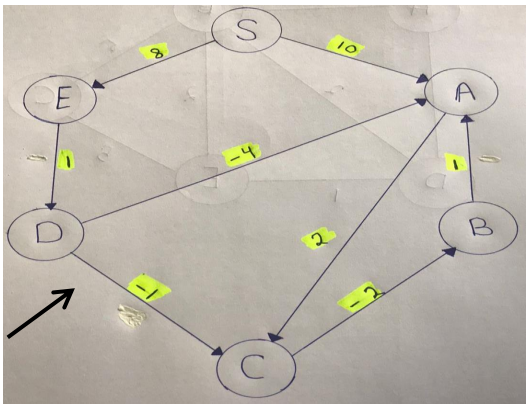


Vertex	Distance
S	0
A	5
B	5
C	7

D	9
E	8

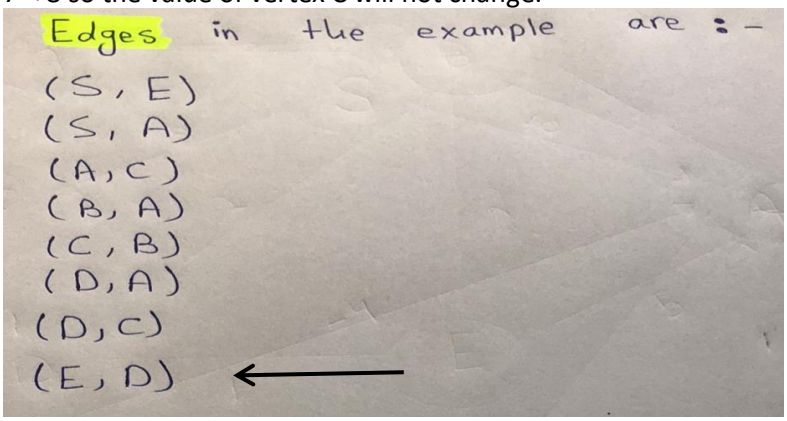
-Sixth edge (D,A), the new distance of vertex A (9-4 =5) and the old distance from the table = 5 , and 5 = 5 so the value of vertex A will not change.

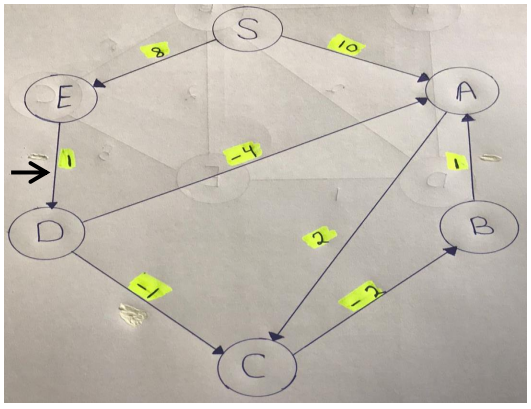




Vertex	Distance
S	0
A	5
B	5
C	7
D	9
E	8

-Seventh edge (D,C), the new distance of vertex C (9-1 =8) and the old distance from the table = 7 , and 7 < 8 so the value of vertex C will not change.

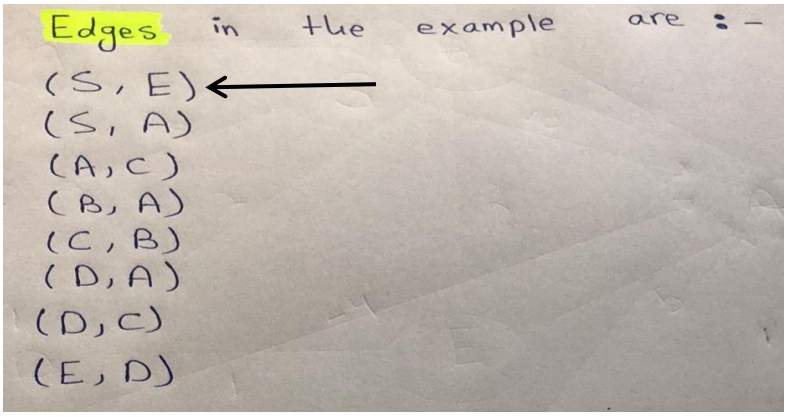


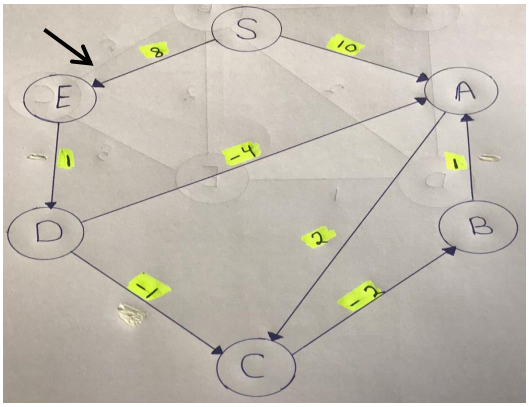


Vertex	Distance
S	0

A	5
B	5
C	7
D	9
E	8

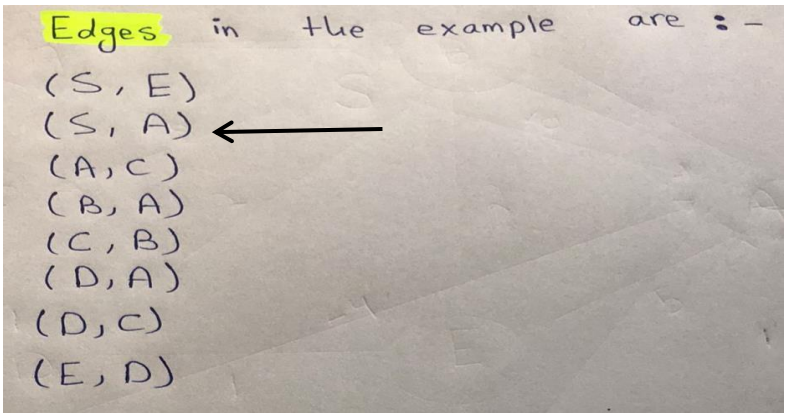
-Eight edge (E,D), the new distance of vertex D (8+1 =9) and the old distance from the table = 9 , and 9 =9 so the value of vertex D will not change.

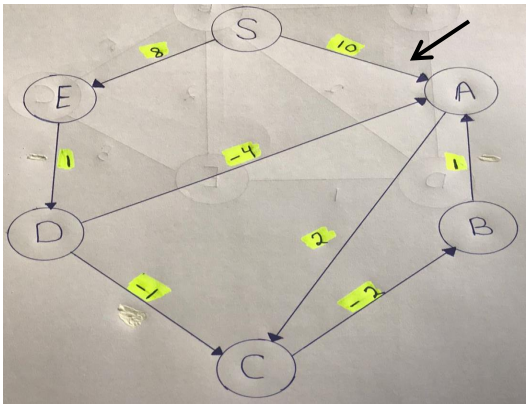




Vertex	Distance
S	0
A	5
B	5
C	7
D	9
E	8

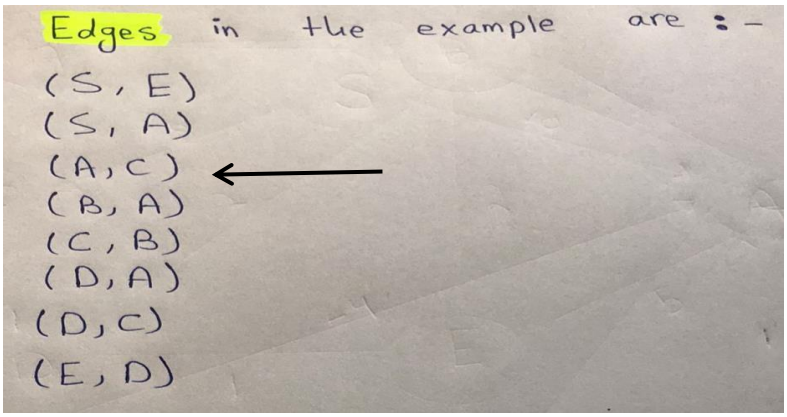
-**Fourth iteration,** in first edge (S,E),the new distance of vertex E (0+8 =8) and the old distance from the table = 8 , and 8=8 so the value of vertex E will not change.

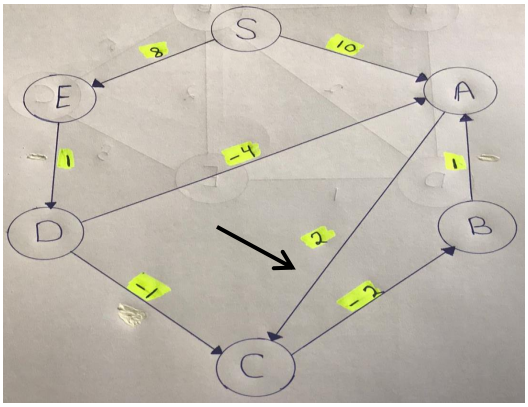




Vertex	Distance
S	0
A	5
B	5
C	7
D	9
E	8

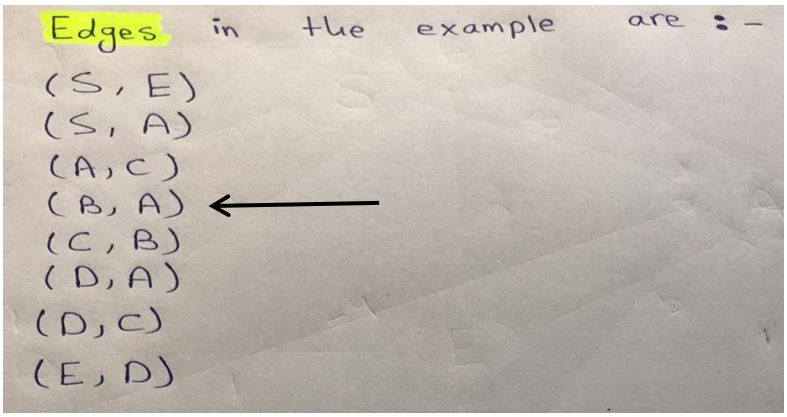
- Second edge(S,A), the new distance of vertex A (0+10 =10) and the old distance from the table = 5 , and 5 < 10 so the value of vertex E will not change.

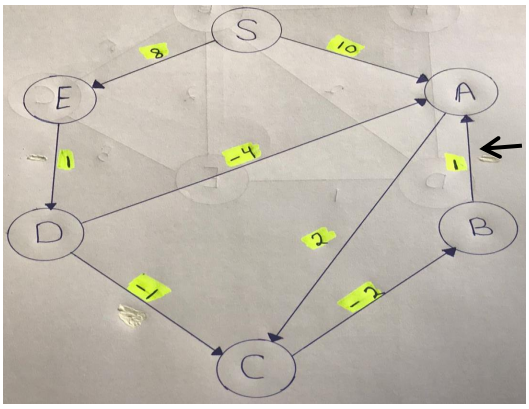




Vertex	Distance
S	0
A	5
B	5
C	7
D	9
E	8

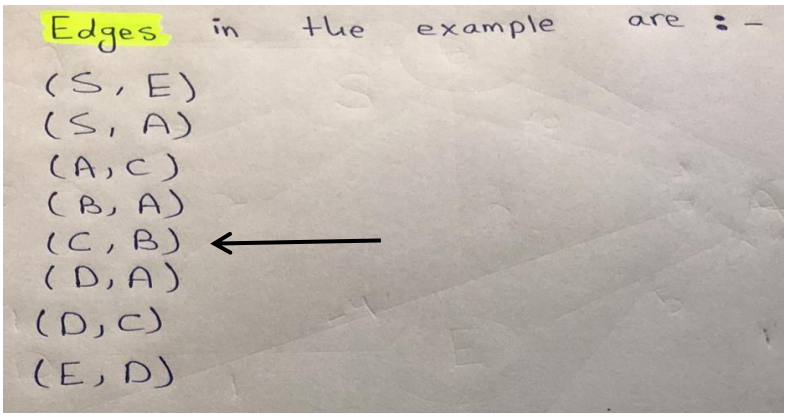
-Third edge(A,C), the new distance of vertex C (5+2 =7) and the old distance from the table = 7 ,
and 7 = 7 so the value of vertex C will not change.

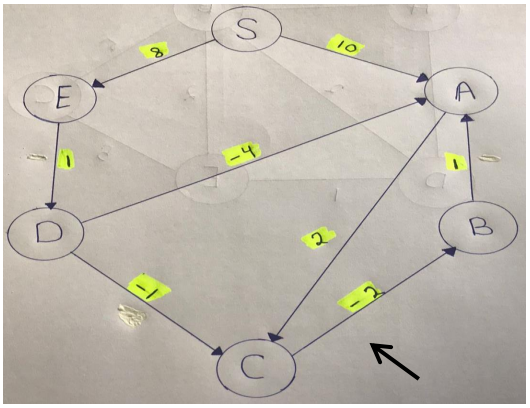




Vertex	Distance
S	0
A	5
B	5
C	7
D	9
E	8

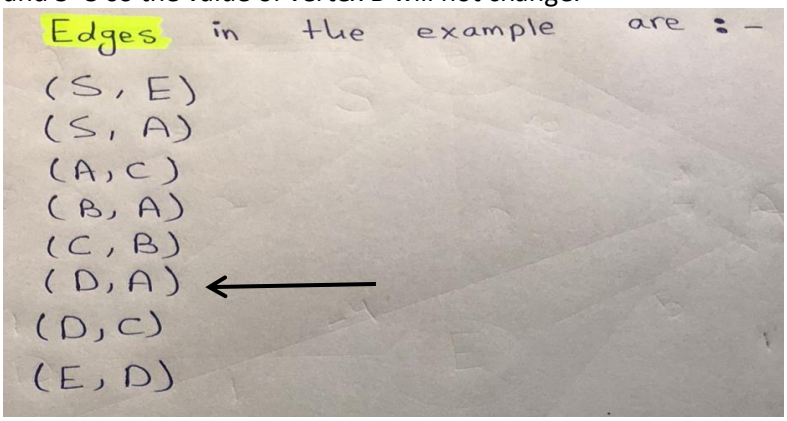
-Fourth edge (B,A), the new distance of vertex A (5+1 =6) and the old distance from the table = 5, and 5<6 so the value of vertex A will not change.

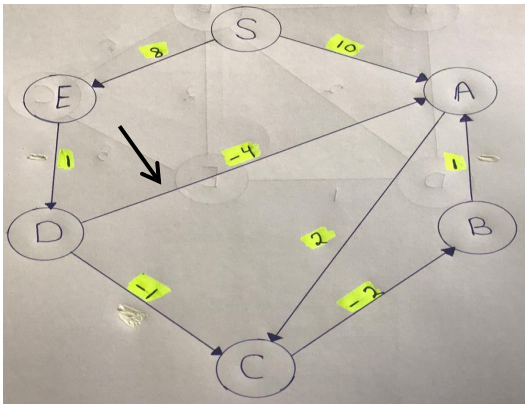




Vertex	Distance
S	0
A	5
B	5
C	7
D	9
E	8

-Fifth edge (C,B), the new distance of vertex B (7-2 =5) and the old distance from the table = 5, and 5=5 so the value of vertex B will not change.

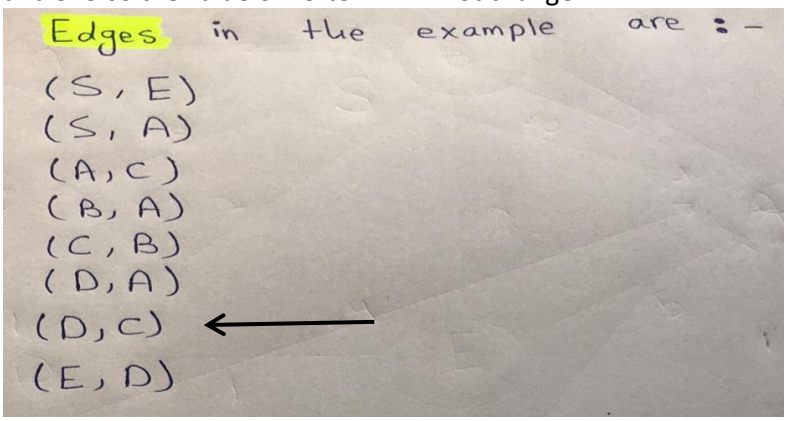


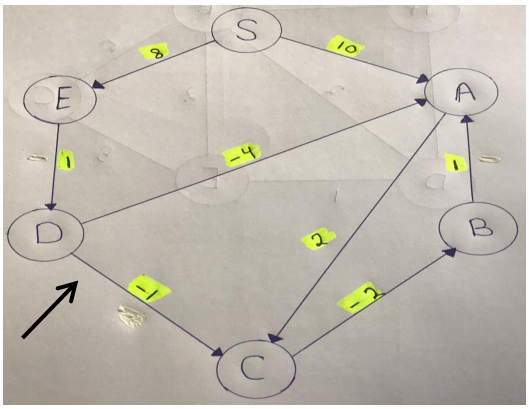


Vertex	Distance
S	0
A	5
B	5
C	7

D	9
E	8

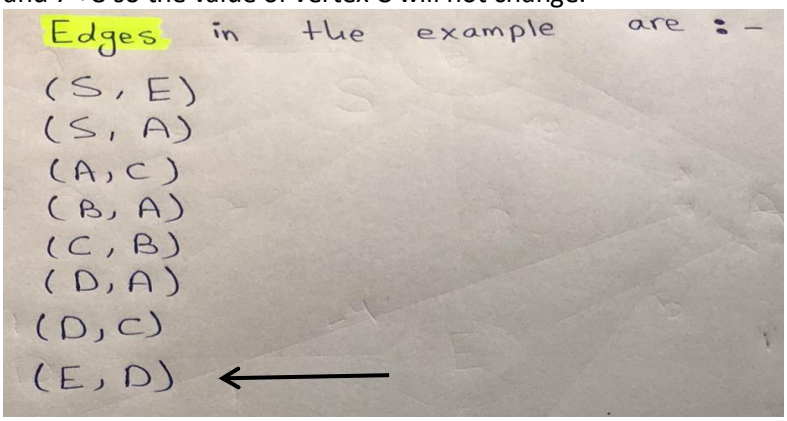
-Sixth edge (D,A), the new distance of vertex A (9-4 =5) and the old distance from the table = 5,
and 5=5 so the value of vertex A will not change.

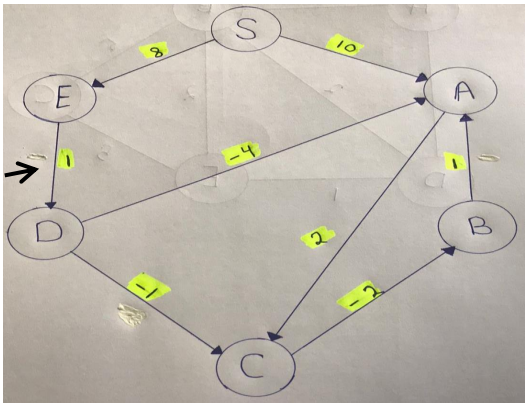




Vertex	Distance
S	0
A	5
B	5
C	7
D	9
E	8

-Seventh edge (D,C), the new distance of vertex C (9-1 =8) and the old distance from the table = 7,
and 7< 8 so the value of vertex C will not change.





Vertex	Distance
S	0

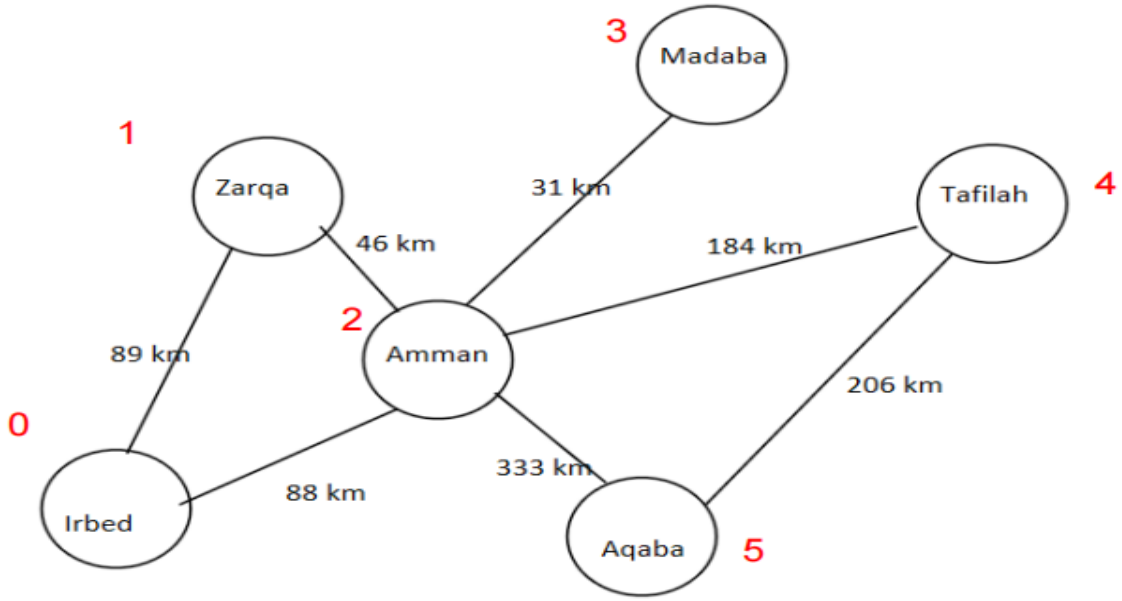
A	5
B	5
C	7
D	9
E	8

-Eight edge (E,D),the new distance of vertex D (8+1 =9) and the old distance from the table = 9,
and 9=9 so the value of vertex D will not change.

** In this iteration, none of the vertices changed the value of the distance, so we will not complete the last iteration and stop in the fourth iteration -> that's how Bellman ford works.

IMPLEMENTATION ON DIJKSTRA'S ALGORITHM

First, this is how I sorted the countries to help me in coding the program:



Code:

```
[16] class Graph():
         def __init__(self, nodes):
             self.nodes = nodes
             self.graph = [[0 for column in range(nodes)]
                           for row in range(nodes)]

       def Final_solutions(graph,distance):
         print("Nodes \t shortest distance")
         for node in range(len(graph)):
           if node==0:
             print("Irbed", "\t\t", distance[node])
           elif node==1:
             print("Zarqa", "\t\t", distance[node])
           elif node==2:
             print("Amman", "\t\t", distance[node])
           elif node==3:
             print("Madaba", "\t\t", distance[node])
           elif node==4:
             print("Tafilah", "\t", distance[node])
           elif node==5:
             print("Aqaba", "\t\t", distance[node])
           else:
             print(node, "\t\t", distance[node])


       def minmum_distance(graph,distance,sptSet):
         minmum = float("inf")
         for count in range(len(graph)):
           if distance[count] < minmum and sptSet[count] == False:
             minmum = distance[count]
             minmum_index = count
```

```
        minmum = distance[count]
        minmum_index = count
    return minmum_index

def dijkstra(graph,start_vertex):
  distance = [float("inf")] * len(graph)
  distance[start_vertex]=0
  sptSet = [False] * len(graph)

  for u in range(len(graph)):
    u= minmum_distance(graph,distance,sptSet)
    sptSet[u]=True

    for v in range(len(graph)):
       if (graph[u][v] > 0 and sptSet[v] == False and distance[v] > (distance[u] + graph[u][v])):
        distance[v] = distance[u] + graph[u][v]

  Final_solutions(graph,distance)
```

Test and see the outcome of the code and find the shortest distance from Irbed to all countries (as an example):

```
g = Graph(6)
g.graph = [[0,89,88,0,0,0],
           [89,0,46,0,0,0],
           [88,46,0,31,184,333],
           [0,0,31,0,0,0],
           [0,0,184,0,0,206],
           [0,0,333,0,206,0]]

test=g.graph
dijkstra(test,0)
```

```
Nodes      shortest distance
Irbed                  0
Zarqa                  89
Amman                  88
Madaba                 119
Tafilah                272
Aqaba                  421
```

ERROR HANDLING IN THE CODES

In code dijkstra's algorithm, I made two handling test:

```
class Graph():
    def __init__(self, nodes):
        self.nodes = nodes

        #error handling
        if(self.nodes<0):
          raise Exception("Not accepting values that less than 0")

        else:
            self.graph = [[0 for column in range(nodes)]
                          for row in range(nodes)]
```

The first error handling, In class graph if the user put the number of nodes that less than (0) and that's impossible, this raise exception will show to him but if he put a number that's grater than zero, the code will run without any problems, to test it let's put a number less than zero and see if the error will show or not:

```
g =Graph(-2)
g.graph = [[0, 89, 88, 0, 0, 0],
           [89, 0, 46, 0, 0, 0],
           [88, 46, 0, 31, 184, 333],
           [0, 0, -4, 0, 0, 0],
           [0, 0, 184, 0, 0, 206],
           [0, 0, 333, 0, 206,1000]]
test=g.graph
dijkstra(test,0)
```

Run the code and the output is:

```
<ipython-input-13-70959e2aa46f> in <module>
----> 1 g =Graph(-2)
      2 g.graph = [[0, 89, 88, 0, 0, 0],
      3              [89, 0, 46, 0, 0, 0],
      4              [88, 46, 0, 31, 184, 333],
      5              [0, 0, -4, 0, 0, 0],

<ipython-input-8-c8bf37af2117> in __init__(self, nodes)
      4         #error handling
      5         if(self.nodes<0):
----> 6           raise Exception("Not accepting values that less than 0")
      7         else:
      8             self.graph = [[0 for column in range(nodes)]

Exception: Not accepting values that less than 0
```

So it work.

Second test handling,In function dijkstra if the user put a number of start vertex is less than zero, the raise exception will show but if the user put a number that is greater than zero, the code will run without any problem, to test let's put a number that is less than zero:

```
    def dijkstra(graph,start_vertex):

      #error handling
      if(start_vertex<0):
        raise Exception("Start source can't be less than 0")

      distance = [float("inf")] * len(graph)
      distance[start_vertex]=0
      sptSet = [False] * len(graph)

      for u in range(len(graph)):
        u= minmum_distance(graph,distance,sptSet)
        sptSet[u]=True

        for v in range(len(graph)):
           if (graph[u][v] > 0 and sptSet[v] == False and distance[v] > (distance[u] + graph[u][v])):
            distance[v] = distance[u] + graph[u][v]

      Final_solutions(graph,distance)
```

```
g =Graph(6)
g.graph = [[0, 89, 88, 0, 0, 0],
           [89, 0, 46, 0, 0, 0],
           [88, 46, 0, 31, 184, 333],
           [0, 0, -4, 0, 0, 0],
           [0, 0, 184, 0, 0, 206],
           [0, 0, 333, 0, 206,1000]]
test=g.graph
dijkstra(test,-3)
```

```
Exception                                 Traceback (most recent call last)
<ipython-input-15-3612e889364a> in <module>
      7            [0, 0, 333, 0, 206,1000]]
      8 test=g.graph
----> 9 dijkstra(test,-3)

<ipython-input-8-c8bf37af2117> in dijkstra(graph, start_vertex)
     43   #error handling
     44   if(start_vertex<0):
---> 45     raise Exception("Start source can't be less than 0")
     46
     47   distance = [float("inf")] * len(graph)

Exception: Start source can't be less than 0
```

In merge sort code, I made just one error handling in the recursive function which is if the user put None in the element in the array, it will error to him, let's test it:

```
[5]  def recursive(Array):

       #error handling
       for i in range(len(Array)):
         if Array[i] == None:
           raise Exception("An element in the array is None, please change it")
```

```
Array=[10,5,3,9,2,4,None]
recursive(Array)
```

Output:

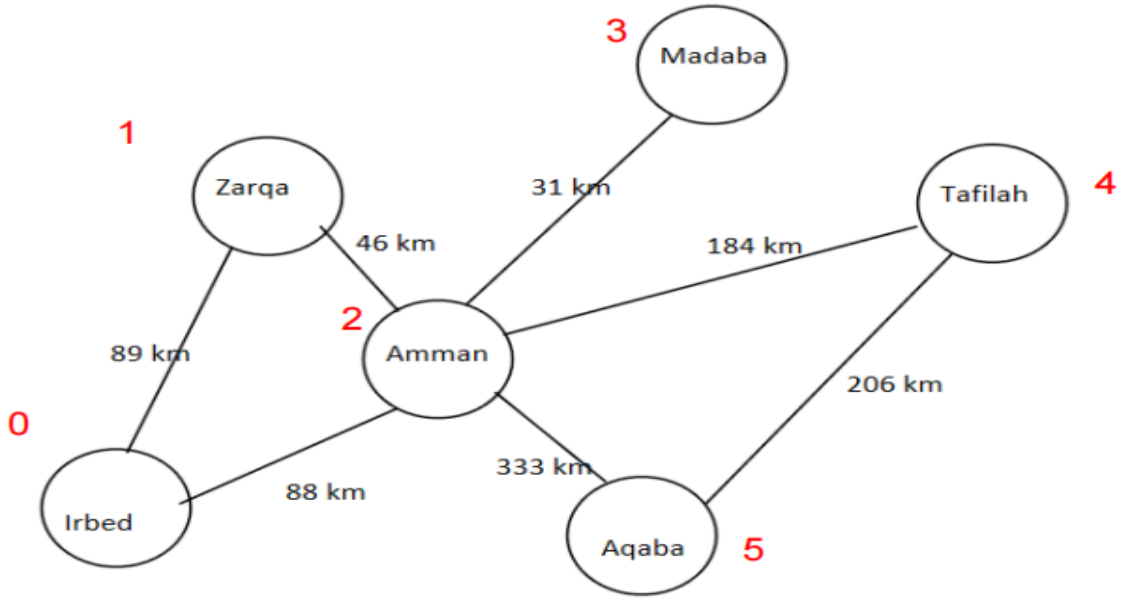
```
Array=[10,5,3,9,2,4,None]
recursive(Array)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-9-826399dcc820> in <module>
      1 Array=[10,5,3,9,2,4,None]
----> 2 recursive(Array)
      3
      4

<ipython-input-5-05ff5b2f8371> in recursive(Array)
      3   for i in range(len(Array)):
      4     if Array[i] == None:
----> 5       raise Exception("An element in the array is None, please change it")
      6
      7   size=len(Array)

Exception: An element in the array is None, please change it
```

All these error handling and with their result may help the program to avoid any glitches and get the best performance, also from raise exception , we can simply identify the error to user clearly without the user needs to search about the problem as shown in the figures, so in conclusion-> the raise helps the program and also the user.

SHORTEST PATH BETWEEN ZARQA AND TAFILAH

To test the code, and see if the code returns the shortest path between the cities in Jordan:



Just a reminder, the figure above is the cities and number of each city.We need to find the shortest path between Zarqa and Tafilah as shown in the figure and see the result.

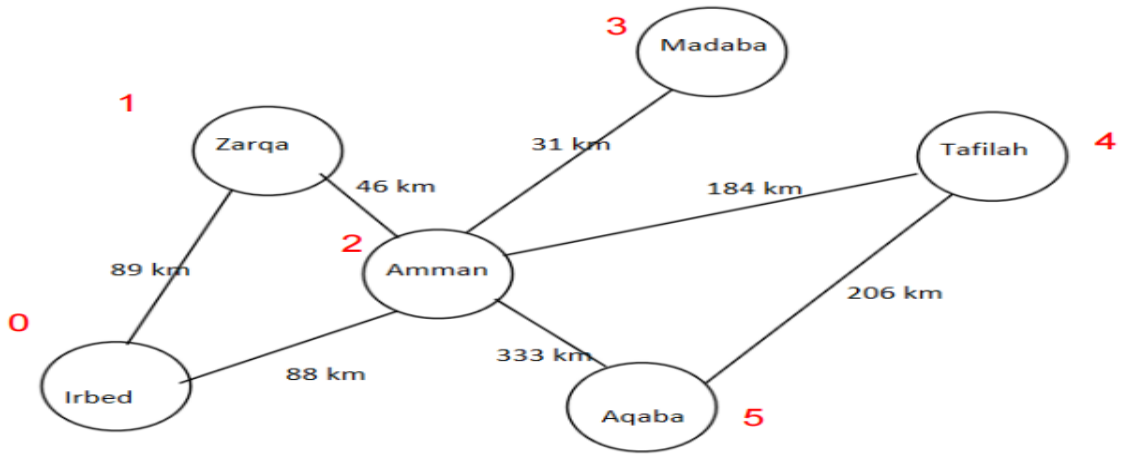
```
g =Graph(6)
g.graph = [[0, 89, 88, 0, 0, 0],
           [89, 0, 46, 0, 0, 0],
           [88, 46, 0, 31, 184, 333],
           [0, 0, -4, 0, 0, 0],
           [0, 0, 184, 0, 0, 206],
           [0, 0, 333, 0, 206,0]]
test=g.graph
dijkstra(test,1)
```

First I called the function graph and set the number of vertices (6), and then put a matrix that contains the path between the cities as shown in the figure, and finally called function dijkstra that return the output(shorted path).

```
g =Graph(6)
g.graph = [[0, 89, 88, 0, 0, 0],
           [89, 0, 46, 0, 0, 0],
           [88, 46, 0, 31, 184, 333],
           [0, 0, -4, 0, 0, 0],
           [0, 0, 184, 0, 0, 206],
           [0, 0, 333, 0, 206,0]]
test=g.graph
dijkstra(test,1)
```

```
Nodes      shortest distance
Irbed                  89
Zarqa                  0
Amman                  46
Madaba                 77
Tafilah                230
Aqaba                  379
```

To check the out put let's back to the figure:



Let's find the shortest path:
Shortest path between Zara -> Tafilah = path between Zara -> Amman + path between Amman->Tafilah.
Shortest path=46+184=230.So the code it worked correctly and returns the shortest path between the two cities.Also from the code , it shows the shortest path from cities to all the cities in the available cities -> from the example above, the code returns the shortest path between Zarqa to all the cities in the figure.

EVALUATE TIME COMPLEXITY ON ALGORITHM

After we write the code, we need to evaluate the efficiency of the algorithm and know the time complexity of it, so first I evaluated the time complexity of class Graph:

```
class Graph():
    def __init__(self, nodes):
        self.nodes = nodes                                  → o(1)

        #error handling
        if(self.nodes<0):
          raise Exception("Not accepting values that less than 0")   → o(1)

        else:
            self.graph = [[0 for column in range(nodes)]    → o(V^2)
                          for row in range(nodes)]
```

As shown in the figure, there is if clause and else so we will pick the worst case:

If clause -> there is only raise exception so the time complexity is O(1) but else-> there are two for loops to put the numbers that the user put in the adjacency matrix so the time complexity is O(V^2) Where (V) is the number of the vertices.
The worst case is the else statement so the time complexity of class graph is O(V^2).

```
def Final_solutions(graph,distance):
  print("Nodes \t shortest distance")          → o(1)
  for node in range(len(graph)):                → o(V)
    if node==0:
      print("Irbed", "\t\t", distance[node])    → o(1)
    elif node==1:
      print("Zarqa", "\t\t", distance[node])    → o(1)
    elif node==2:
      print("Amman", "\t\t", distance[node])    → o(1)
    elif node==3:
      print("Madaba", "\t\t", distance[node])   → o(1)
    elif node==4:
      print("Tafilah", "\t", distance[node])    → o(1)
    elif node==5:
      print("Aqaba", "\t\t", distance[node])    → o(1)
    else:
      print(node, "\t\t", distance[node])       → o(1)
```

Function final solution is to print the solution of the shortest path: the first print inside the function is O(1) and after print there is a for loop so the time complexity : T =O(1)+V*O(1) -> T= O(V) WHERE V is the number of vertices from the graph.

```
def minmum_distance(graph,distance,sptSet):
  minmum = float("inf")                          o(1)
  for count in range(len(graph)):                o(V)
    if distance[count] < minmum and sptSet[count] == False:
      minmum = distance[count]                   o(1)
      minmum_index = count                       o(1)
  return minmum_index                            o(1)
```

Function minimum distance is to find the shortest path: there is a for loop so the time complexity :
T =O(1)+V*O(1) -> T= O(V) WHERE V is the number of vertices from the graph.

```
def dijkstra(graph,start_vertex):

  #error handling
  if(start_vertex<0):
    raise Exception("Start source can't be less than 0")        o(1)

  distance = [float("inf")] * len(graph)                        o(1)
  distance[start_vertex]=0                                      o(1)
  sptSet = [False] * len(graph)                                 o(1)

  for u in range(len(graph)):
    u= minmum_distance(graph,distance,sptSet)                   v*o(v)
    sptSet[u]=True                                              v*o(1)

    for v in range(len(graph)):
       if (graph[u][v] > 0 and sptSet[v] == False and distance[v] > (distance[u] + graph[u][v])):
        distance[v] = distance[u] + graph[u][v]                 v^2*o(1)

  Final_solutions(graph,distance)                               o(v)
```

Function dijkstra is to the final function in this code and the time complexity :
T =O(1)+O(1)+O(1)+O(1)+V*O(V)+V*O(1)+V^2*O(1)+O(V) -> we will take the worst case
T= O(V^2) is the time complexity of the dijkstra algorithm, WHERE V is the number of vertices from the graph.

References

PART one :

C)

(Asymptotic Analysis (Based on input size) in Complexity Analysis of Algorithms - GeeksforGeeks, 2022)

D)

Faceprep.in. 2022. FACE Prep | Land your dream Tech job with FACE Prep. [online] Available at: <https://www.faceprep.in/data-structures/space-complexity/> [Accessed 1 September 2022].

E)

GeeksforGeeks. 2022. Time-Space Trade-Off in Algorithms - GeeksforGeeks. [online] Available at: <https://www.geeksforgeeks.org/time-space-trade-off-in-algorithms/> [Accessed 1 September 2022].

F)

Assignmentexpert.com. 2022. Answer in Java | JSP | JSF for Rakan #149033. [online] Available at: <https://www.assignmentexpert.com/homework-answers/programming-and-computer-science/java-jsp-jsf/question-149033> [Accessed 1 September 2022].

K)

ThoughtCo. 2022. What Is Encapsulation ?. [online] Available at: <https://www.thoughtco.com/definition-of-encapsulation-958068> [Accessed 1 September 2022].

Techopedia.com. 2022. What is Information Hiding? - Definition from Techopedia. [online] Available at: <https://www.techopedia.com/definition/3814/information-hiding> [Accessed 1 September 2022].

I)

Quora. 2022. What is imperative ADT? Is abstract data type in data structure similar to abstraction in object orientation?. [online] Available at: <https://www.quora.com/What-is-imperative-ADT-Is-abstract-data-type-in-data-structure-similar-to-abstraction-in-object-orientation> [Accessed 1 September 2022].

Cs.auckland.ac.nz. 2022. Data Structures and Algorithms: Objects and ADTs. [online] Available at: <https://www.cs.auckland.ac.nz/software/AlgAnim/objects.html> [Accessed 1 September 2022].