Algorithms and Data Structures

09/11/2021lec2

1. What is an algorithm?

   Organized set of commands given to an any processing machine to execute certain task.

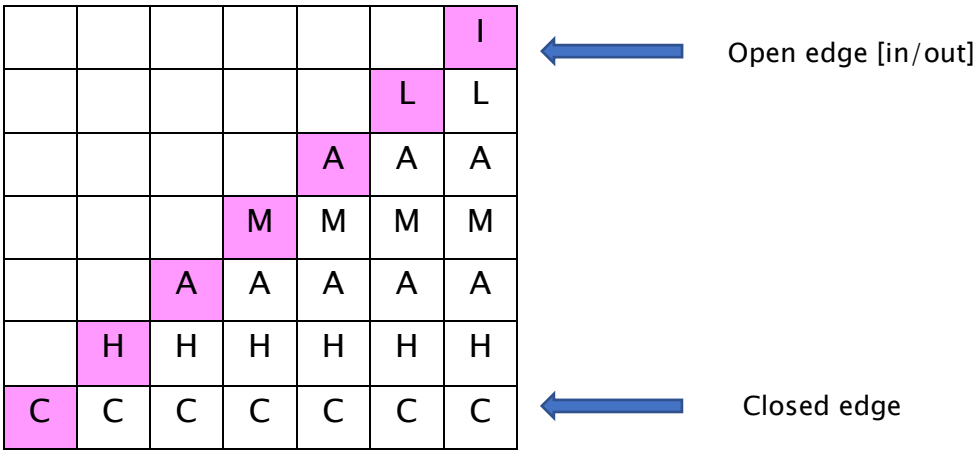
2. What's a structure?

   Structure is a user defined data type available in C that allows to combine items of different kinds.

Stack

- Storing data one over another in the column format refers to stack concept.
- Stack allows us to access only one data at a time.
- Inside a stack, the element that we insert last will be the first to take out.

Storing my name inside a stack graphical representation



I						
L	L					
A	A	A				
M	M	M	M			
A	A	A	A	A		
H	H	H	H	H	H	
C	C	C	C	C	C	C

Q] Saman wants to keep track of his cars in sale. He might want to track the following attributes about each car.

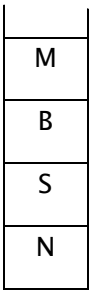
→ Make
→ Model
→ Chasi number
→ Engine capacity

```
#include <stdio.h>
#include <string.h>

struct cars
{
    char make[20];
    char model[20];
    char chasino[20];
    float engcpy;
};

int main()
{
    struct cars c1;
    struct cars c2;

    strcpy(c1.make, "Nissan");
    strcpy(c1.model, "Leaf");
    strcpy(c1.chasino,
"NKL45632");
    c1.engcpy=1500;

    printf("%s\n", c1.make);
    printf("%s\n", c1.model);
    printf("%s\n", c1.chasino);
    printf("%f\n", c1.engcpy);
}
```

16/11/2021lec3

Implementing an integer stack size 10

- Since the stack has only one end it requires only one pointing variable.
- In the beginning when the stack is empty pointer variable pointing to index -1.
- When we're inserting value pointer will increment. (Pointer++)
- When we're withdrawing values pointer will decrement.



Open

Close

Inserting values into the stack. PUSH()

Check whether stack is full or not.

If it's not full

STEP 1: increase the top by 1

STEP 2: assign the value into the cell pointing by top.

Else

Output can't add values stack is full

Removing values into the stack. POP()

- Check whether stack is empty or not.
- STEP 1: remove the value pointing by top.
- STEP 2: decrement top by 1.

Checking the stack is empty or not. ISEMPTY()

→ If top == -1 → stack is empty.

Checking the stack is full or not. ISFULL()

→ If top == size -1 → stack is full.

Q] find out the applications of stack in computing?

Exception handling

Memory management

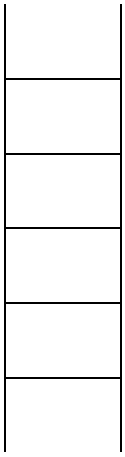
```
#include <stdio.h>
int stack[10]; //implementing the stack
int top = -1;
int size = 10;
void push(int data)
{
   if(isfull()==0)
   {
      top=top+1; //insertion of values
   stack[top]=data;
   }
   else
      printf("Can't add more values due to stack is full\n");
}
int pop()
{
   if(isempty()==0)
   {
      int temp;
      temp = stack[top];
      top = top-1;
      return temp;
   }
   else
      printf("Can't extract values. stack is empty\n");
}
int isempty()
{
   if(top==-1)
      return 1; //stack is empty it will return 1
   else
      return 0;
}
int isfull()
{
    if(top== size-1)
      return 1; //stack is full it will return 1
   else
      return 0;

}
void printdata()
{
   int i;
   printf("Stack contents \n");
   for(i=top;i>=0;i--)
   {
      printf("%d\n",stack[i]);
   }
}
int main()
{
   push(56);
   push(13);
   push(78);
   push(156);
   push(103);
   push(72);
   push(596);
   push(130);
   push(777);
   push(6);
   push(60);
   printdata();
   //printf("Removed value is %d\n",pop());
   //printf("Removed value is %d\n",pop());
   //printf("Removed value is %d\n",pop());
   //printf("Removed value is %d\n",pop());//no data to remove
   //printdata();
}
```

23/11/2021lec4

Queue

- Queues consider as a linear data structure which storing data in contiguous cells according to first in first out method [FIFO].
- The first element to be removed from the queue is the first element we inserted to the queue.

Structure of a queue



Inserting values
[open edge]

Taking out values
[open edge]

Q] Inserting my name into a queue graphical representation

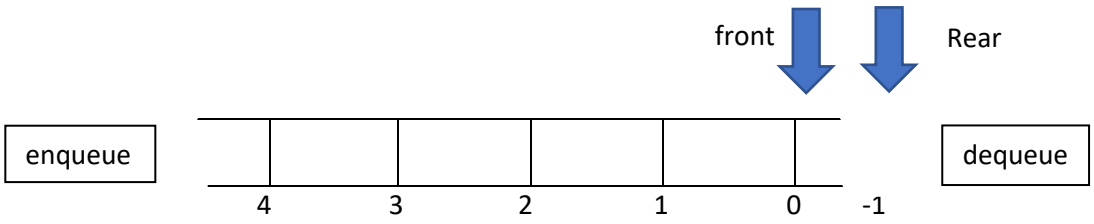
						I
					L	L
				A	A	A
			M	M	M	M
		A	A	A	A	A
	H	H	H	H	H	H
C	C	C	C	C	C	C

I	I	I	I	I	I	I
L	L	L	L	L	L	
A	A	A	A	A		
M	M	M	M			
A	A	A				
H	H					
C						

Implementation of queue

- ♥ Queue has two open edges. Therefore, we need two pointing variables.
  - → Front 0
  - → Rear -1

- ♥ Rear pointer responsible for enqueue (insert) operation.
- ♥ Front pointer is responsible for dequeue operation.



1) Enqueue ( )
   I. Check if the queue is full or not
      - If its full → can't add more values
      - If not full → let's add values
   II. Rear needs to be increment by one.
   III. The data will occupy that index cell.

2) Dequeue ( )
   I. Check if the queue is empty or not.
      - If it's empty → can't dequeue more values.
      - If not empty → let's dequeue values.
   II. Remove the value point by front.
   III. Front needs to be increment by one.

3) Isfull ( )

   If the pointer rear == maximum size of the queue → full

   Else

   Queue → is not full

4) Isempty ( )

For a queue to be empty there are two conditions that can happen.

1. If rear == -1 queue is empty.
2. If the front overrides rear again queue is empty

```
#include <stdio.h>

int queue[5];
int max=5;
int front=0;
int rear=-1;

void enqueue(int data){
if(isfull()==0){
rear=rear+1;
queue[rear]=data;
}

 else{
printf("cannot add more vales Queue is full !!\n");
}
}

void dequeue(){
if(isempty()==0){
printf("Value extracted is %d\n",queue[front]);
front = front+1;
}

 else
printf("can't dequeue queue is empty !!\n");

}

int isfull(){
if(rear==max-1){
//printf("Queue is full !!\n");
return 1;
}
else
//printf("Queue is not full !!\n");
return 0;
}

int isempty(){
if(rear==-1 || front>rear)
return 1;
else
return 0;
}

void print(){
int i;
printf("The Queue\n");
for(i=front;i<=rear;i++){
printf("%d\n",queue[i]);
}
}
int main()
{
enqueue(25);
enqueue(75);
dequeue();
dequeue();
dequeue();

return 0;
}
```

30/11/2021lec5

What is an algorithm?

- Series of text breakdown from a specific task that carries out of overcome from a problem.

Linear search	Binary search
Small arrays	Large arrays
Unsorted data	Sorted data

Linear search

- ♥ Start at first element of array.
- ♥ Compare value to value (key) for which you are searching.
- ♥ Continue with next element of the array until you find a match or reach the last element in the array.
- ♥ Note: on the average we have to compare the search key with half the elements in the array.

Ann	Keil	John	Bob	Mary	Jennie			
0	1	2	3	4	5	6	7	

Characteristics of a linear search

- ♥ Data inside the array no need to be sorted.
- ♥ It's easy to implement.
- ♥ Not suitable for large data sets.
- ♥
  - → Best case: the element that we're searching is at the start.
  - → Worst case: the element that we're searching is at the end.

Q] find 53.

Iteration 1	12	85	-6	0	36	74	62	53	11	9	
Iteration 2	12	85	-6	0	36	74	62	53	11	9	
Iteration 3	12	85	-6	0	36	74	62	53	11	9	
Iteration 4	12	85	-6	0	36	74	62	53	11	9	
Iteration 5	12	85	-6	0	36	74	62	53	11	9	
Iteration 6	12	85	-6	0	36	74	62	53	11	9	
Iteration 7	12	85	-6	0	36	74	62	53	11	9	
Iteration 8	12	85	-6	0	36	74	62	53	11	9	**Found at position 7**

CODE

<u>Linear search flow chart</u>

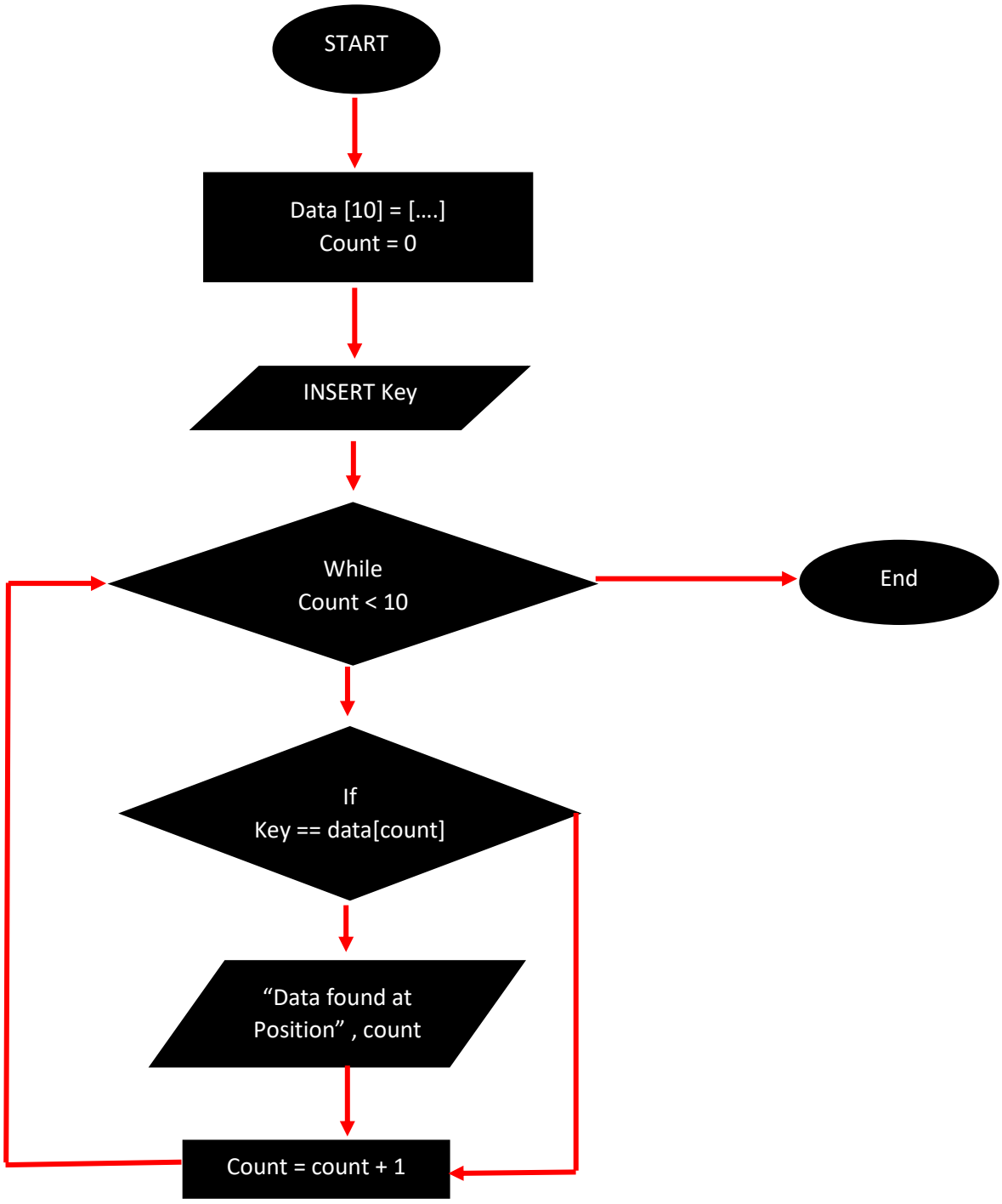
Declare dataset [10] = [….]

Insert key

Set a loop for 10 times

Check key ==data [position] ?

```
#include <stdio.h>
int main()
{
   int data[10]={45,89,63,-3,34,74,11,25,100,98};
   int count=0,key;
   int found=0;

   printf("enter number to search\n");
   scanf("%d", &key);

   while(count<10 && found==0 )
   {
      if(key==data[count]){
         printf("Data is found at position %d\n",count);
         found=1;
      }
      count=count+1;
   }
   printf("count value %d\n",count);

   if(found==0)
      printf("Data is not in the list!");
}
```


START
Data [10] = [….]
Count = 0
INSERT Key
While
Count < 10
End
If
Key == data[count]
“Data found at
Position” , count
Count = count + 1

Q] Complete the following trace table for linear search algorithm with following data.

Data = {45, 89, 63, -3, 43, 74, 11, 25, 100, 98}

Key → 25

count	While count<10	If key == data [count]	output	Count = count +1
0	0 < 10 → true	25 == 45 → false	-	0 + 1
1	1 < 10 → true	25 == 89 → false	-	1 + 1
2	2 < 10 → true	25 == 63 → false	-	2+ 1
3	3 < 10 → true	25 == -3 → false	-	3 + 1
4	4 < 10 → true	25 == 43 → false	-	4 + 1
5	5 < 10 → true	25 == 74 → false	-	5 + 1
6	6 < 10 → true	25 == 11 → false	-	6 + 1
7	7 < 10 → true	25 == 25 → true	7	7 + 1
8	8 < 10 → true	25 == 100 → false	-	8 + 1
9	9 < 10 → true	25 == 100 → false	-	9 + 1
10	10 < 10 → false	-	-	-

07/12/2021lec6

Binary search algorithm

- Binary search only works for sorted data set.
- It starts searching numbers always from the middle of the array.
- If the value reached in the middle of the array position found. Otherwise, it need to decide which half of the array needs to consider and ignores the other half.

Graphical representation of binary search

Q] find the value 78

9	17	21	34	39	42	51	55	63	67	78	79

Mid = (0 + 11)/2 = 5

Is found 78 == 42 → false

Is it < 42 → false

Is it > 42 (current mid) → true

51	55	63	67	78	79

Mid = (6 + 11)/2 = 8

Is found 78 == 63 → false

Is it < 63 → false

Is it > 63 (current mid) → true

67	78	79

Mid = (9 + 11)/2 = 10

Is found 78 == 78 → true

[we found it at index 10]

Q] find the value 17

9	17	21	34	39	42	51	55	63	67	78	79

Mid = (0 + 11)/2 = 5

Is found 17 == 42 → false

Is it < 42 → true

9	17	21	34	39

Mid = (0 + 4)/2 = 2

Is found 17 == 21 → false

Is it < 21 → true

9	17

Mid = (0 + 1)/2 = 0

Is found 17 ==9 → false

Is it < 17 → false

Is it > 17 → true

17

Mid = (0 + 0)/2 = 0

Is found 17 == 17 → true

[we found it at index 1]

Binary search implementation

Key components

Mid – (first index + last index)/2

If data [mid] == key?

If key < data[mid] ---

Last = mid – 1

If key > data[mid] ---

First = mid + 1

Sorting algorithms

Selection sort is one of the most simplest sorting algorithms.

Concept for sort in ascending order :

- ♥ Locate smallest element in array. Exchange it with element in position o.
- ♥ Locate next smallest element in array. Exchange it with element in position 1.
- ♥ Continue until all elements are arranged in order.

17	6	15	4	0	12	-2
-2	6	15	4	0	12	17
-2	0	15	4	6	12	17
-2	0	4	15	6	12	17
-2	0	4	6	15	12	17
-2	0	4	6	12	15	17

According to the above diagram,

1) Per each iteration it is selecting current position to replace with the minimum value.
2) Selecting the minimum value from the unsorted regen.
3) Exchanging it with the current position.

Swapping variables

```
int x=15;
int y=50;
int temp;
temp=x;
x=y;
y=temp;

printf("%d",x)---→ 50
printf("%d",y)---→ 15
```

Selection sort implementation

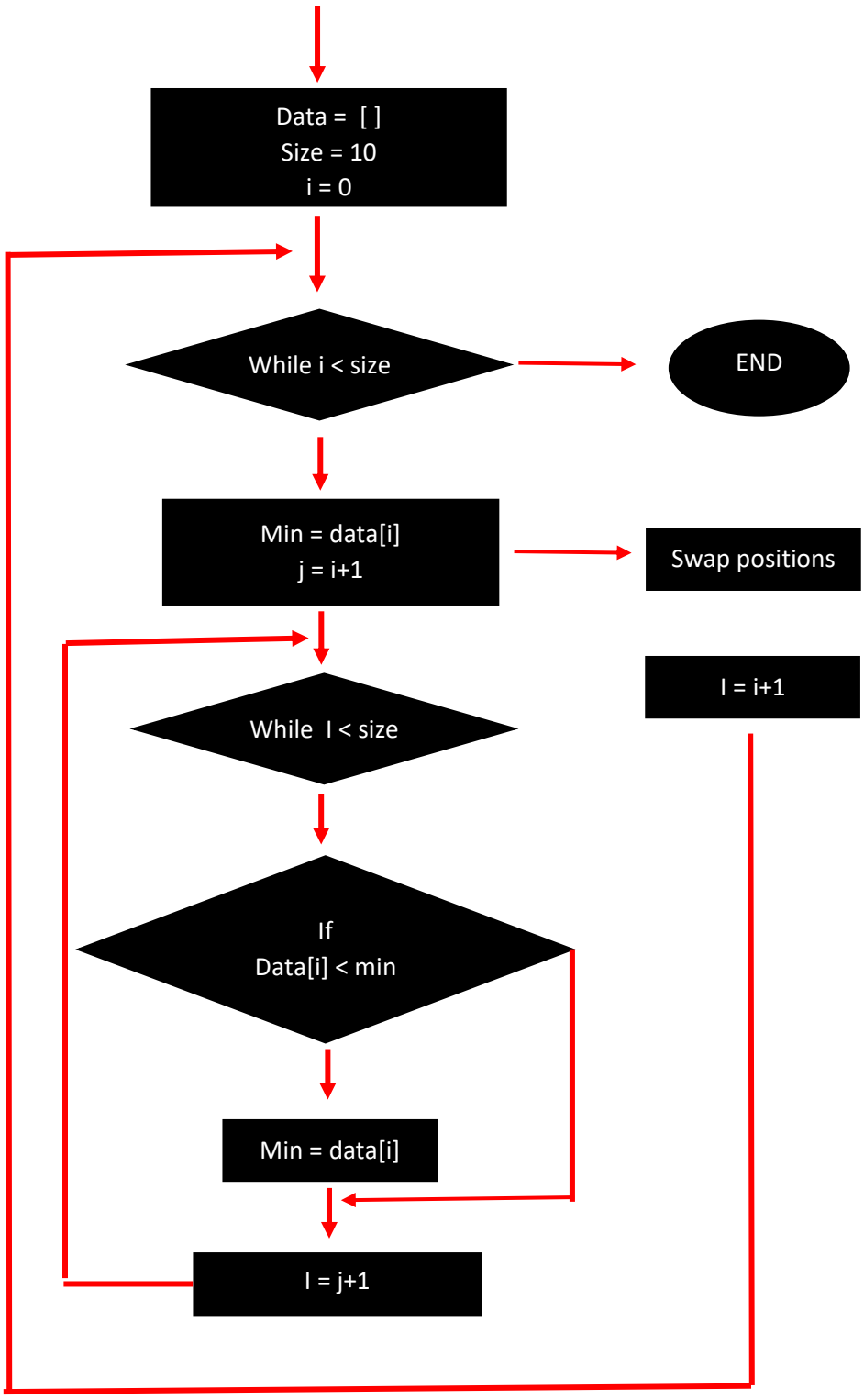
12	85	3	98	-4	0	7	100	10	31
-4	85	3	98	12	0	7	100	10	31
-4	0	3	98	12	85	7	100	10	31
-4	0	3	98	12	85	7	100	10	31
-4	0	3	7	12	85	98	100	10	31
-4	0	3	7	10	85	98	100	12	31
-4	0	3	7	10	12	98	100	85	31
-4	0	3	7	10	12	31	100	85	98
-4	0	3	7	10	12	31	85	100	98
-4	0	3	7	10	12	31	85	98	100

For the selection sort, we need two loops.

1. Locate the next starting minimum.

2. Locate the actual minimum from the remaining data.


```
#include <stdio.h>
int main()
{
   int data[5] = {45, -9, 23, 0, 5};
   int i,j,minpos,size=5,min,temp;
   for(i=0;i<(size-1);i++)
   {
      min=data[i];
      minpos=i;
      for(j=i+1;j<size;j++)
      {
         if(data[j]<min)
         {
            min=data[j];
            minpos=j;
         }
      }
         temp = data[i];
         data[i]=data[minpos];
         data[minpos]=temp;
      }

   for(i=0;i<size;i++)
   {
      printf("%d\t",data[i]);
   }
}
```


Data = [ ]
Size = 10
i = 0
While i < size
END
Min = data[i]
j = i+1
Swap positions
I = i+1
While I < size
If
Data[i] < min
Min = data[i]
I = j+1

Bubble sort

- Bubble sort compares two adjacent values of an array each time an if it is not in order values will be exchange.
- Continuing the same process until the end of array.
- If array is not fully sorted continuing the above process again and again until it sorted.

25	26	16	18	15
25	26	16	18	15
25	16	26	18	15
25	16	18	26	15
25	16	18	15	26

PASS 1

25	16	18	15	26
16	25	18	15	26
16	18	25	15	26
16	18	15	25	26
16	18	15	25	26

PASS 2

16	18	15	25	26
16	18	15	25	26
16	15	18	25	26
16	15	18	25	26
16	15	18	25	26

PASS 3

16	15	18	25	26
15	16	18	25	26

PASS 4

For the implementation

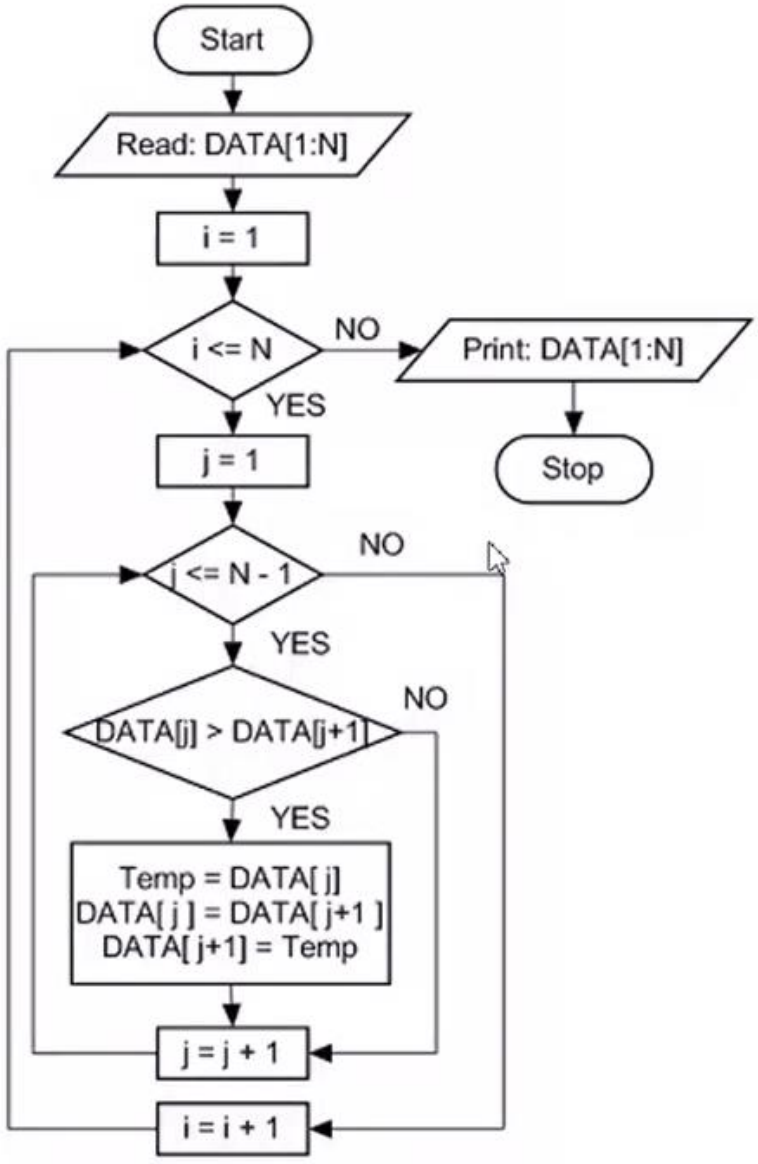
Need to compare pairs until the end of array

If array size is n → (n−1) compares

COMPARES

Continue the above process <n times

PASSES



Bubble Sort Flowchart

```
#include <stdio.h>
int main()
{
   int data[5]={9,3,0,5,2};
   int i,j,temp,size=5,pass=0;
   for(i=0;i<size;i++)
   {
      for(j=0;j<size-1;j++)
      {
         if(data[j]>data[j+1])
         {
         temp=data[j];
         data[j]=data[j+1];
         data[j+1]=temp;
         }
      }
      pass=pass+1;
   }
   for(i=0;i<size;i++)
   {
      printf("%d\t",data[i]);
   }
   printf("\nNumber of pases %d\n",pass);
}
```

```
#include <stdio.h>
int main()
{
   int data[5]={3,2,4,5,6};
   int i,j,temp,size=5,pass=0,ex=1;

   while(i<size && ex!=0)
   {
      ex=0;
      for(j=0;j<size-1;j++)
      {
         if(data[j]>data[j+1])
         {
         temp=data[j];
         data[j]=data[j+1];
         data[j+1]=temp;
         ex=ex+1;
         }
      }
      pass=pass+1;
      i=i+1;
   }
   for(i=0;i<size;i++)
   {
      printf("%d\t",data[i]);
   }
   printf("\nNumber of pases %d\n",pass);
}
```

18/01/2022lec10

Insertion sort

- It divides the dataset into two regions. (Sorted and unsorted)
- From element to element select elements from unsorted regions and inserting it to the exact correct position in the sorted region.

31	9	-7	6	10	5	50	-13
9	31	-7	6	10	5	50	-13
-7	9	31	6	10	5	50	-13
-7	6	9	31	10	5	50	-13
-7	6	9	10	31	5	50	-13
-7	5	6	9	10	31	50	-13
-7	5	6	9	10	31	50	-13
-13	-7	5	6	9	10	31	50

15	16	6	8	12	5
15	16	6	8	12	5
6	15	16	8	12	5
6	8	15	16	12	5
6	8	12	15	16	5
5	6	8	12	15	16

Implementation of insertion sort

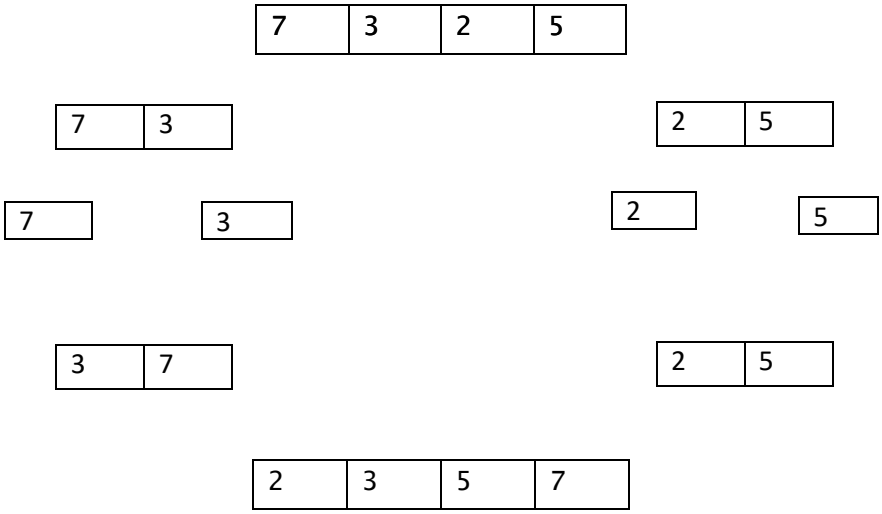
- Need a loop to start from second element and to reach last element one by one.
- Now have to compare this value with the elements in the sorted region.
  - → If data < unsorted value → increment position by one
  - → If not stop the comparison and place it where it stopped.

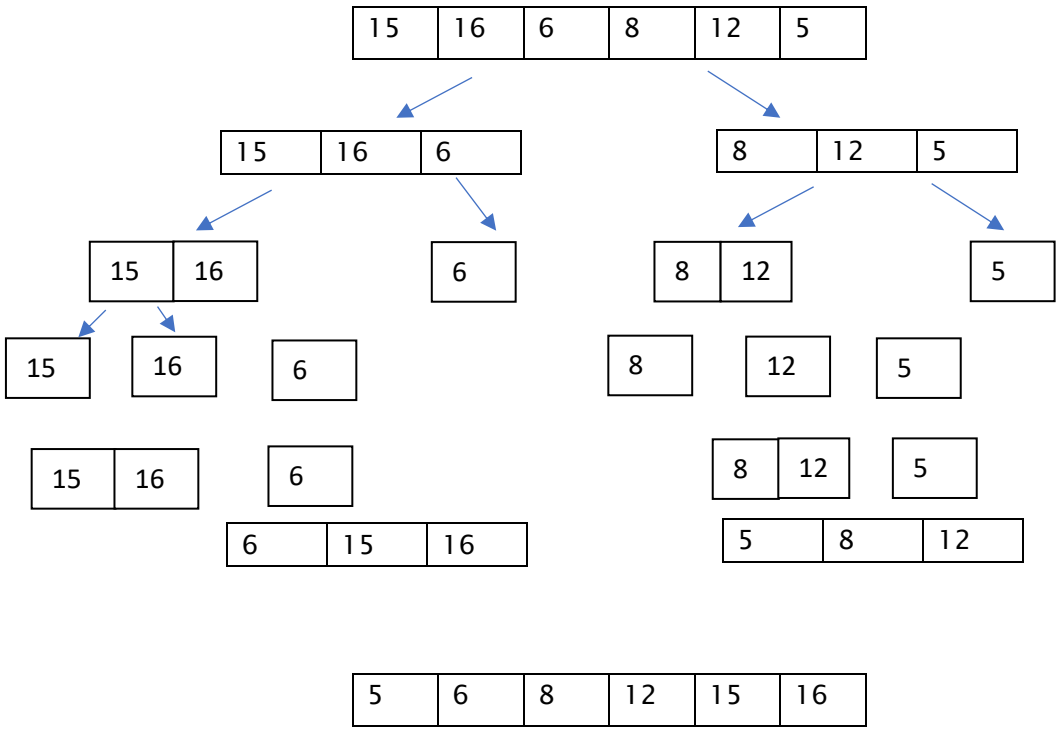
```
#include <stdio.h>
int main()
{
   int data[5]={6,3,7,12,10};
   int i,j,key;

   for(i=1;i<5;i++)
   {
      j=i-1;
      key=data[i];

      while(j>=0 && data[j]>key)
      {
         data[j+1]=data[j];
         j=j-1;
      }
      data[j+1]=key;
   }
   for(i=0;i<5;i++)
   {
      printf("%d\t",data[i]);
   }
}
```

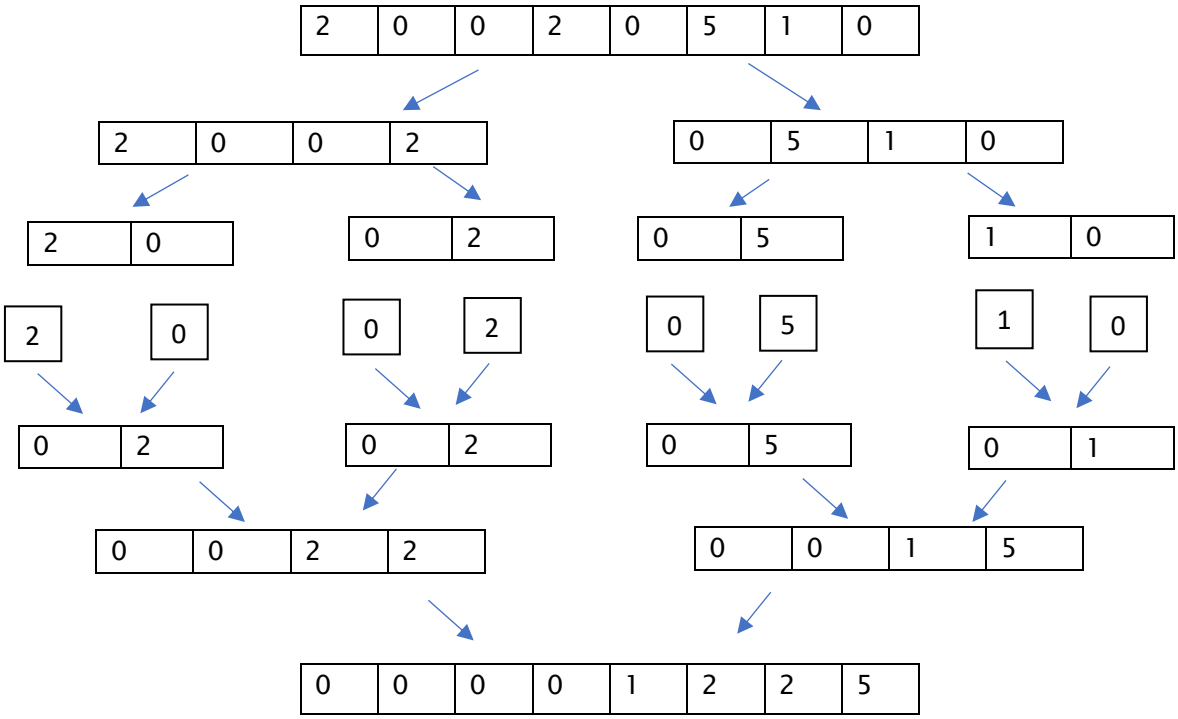
Merge sort

In merge sort the algorithm breaking down the given dataset into two subsets per each iteration until each element is an isolated subset.

Now comparing elements by adjacent pairs. And sorting them in the correct order and merge these subsets until it creates the entire dataset.







Algorithmic analysis

What is a good algorithm?

- Efficient depends on running time and space used.
- Efficiency as a function of input size:
  - → The number of bits in an input number
  - → Number of data elements (number, points)

```
#include <time.h>
#include <stdio.h>

int main()
{
   clock_t start,end,t;
   start = clock();
   int i;
   for(i=0;i<100;i++)
   {
      printf("hello world\n");
   }
   t = clock() - t;

   double time_taken =
((double)t)/CLOCKS_PER_SEC;

   printf("took %f seconds to execute
\n", time_taken);
}
```

Data structures

- Stacks
- Queue
- List

Algorithms

- Searching
  - → Binary search
  - → Linear search
- Sorting
  - → Bubble sort
  - → Selection sort
  - → Insertion sort
  - → Merge sort

Definition	
Stack	Storing data one over another in the column format.
Queue	a linear data structure which storing data in contiguous cells according to FIFO.
Linear search	A method for finding an element within a list of unsorted data.
Binary search	A method for finding an element within a list of sorted data.
Selection sort	Is an in-place comparison sorting algorithm. (swapping)
Bubble sort	compares two adjacent values of an array each time an if it is not in order values will be exchange.
Insertion sort	It divides the dataset into two regions. (Sorted and unsorted)
Merge sort	the algorithm breaking down the given dataset into two subsets per each iteration until each element is an isolated subset.

01/02/2022lec11

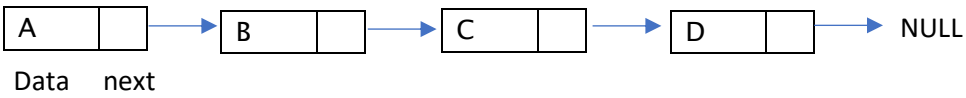
Linked lists

A linked list is a sequence of data structures, which are connected together via links. Linked list is a sequence of links which contains data. It contains a connection to another link. Also, it's the second most-used data structure after array.

Link – each link of a linked list can store a data called an element.

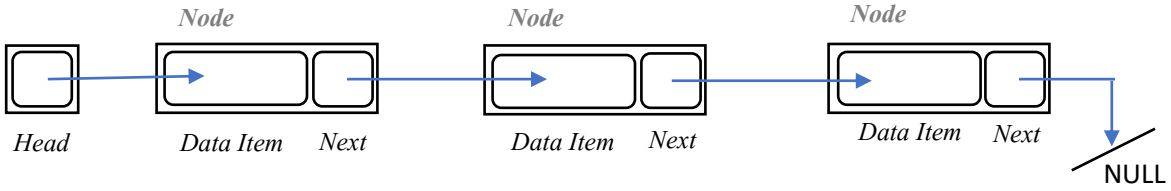
Next – each link of a linked list contains a link to the next link called next.

LinkedList – a linked list contains the connection link to the first link called first



Graphical representation

Linked list can be visualized as a chain of nodes, where every node points to the next node.



As per the above illustration, following are the important points to be considered.

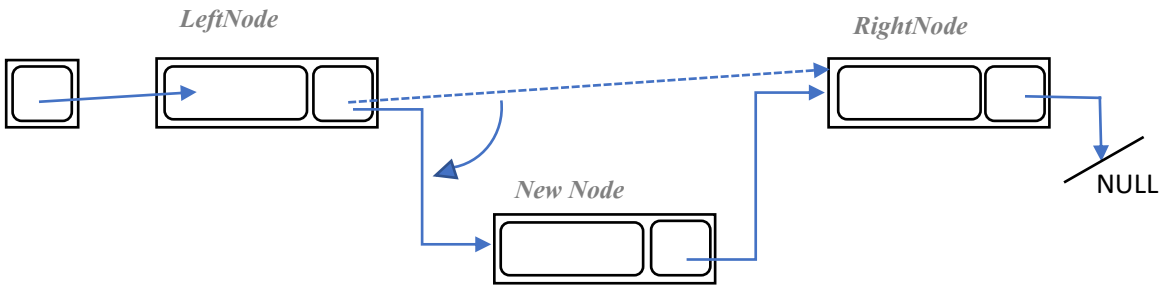
- Linked list contains a link element called first/ head.
- Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.

Insertion

Adding a new node in linked list is a more than one step activity.
Refers to the diagram,
First create a node using the same structure and find the location where is must be inserted.

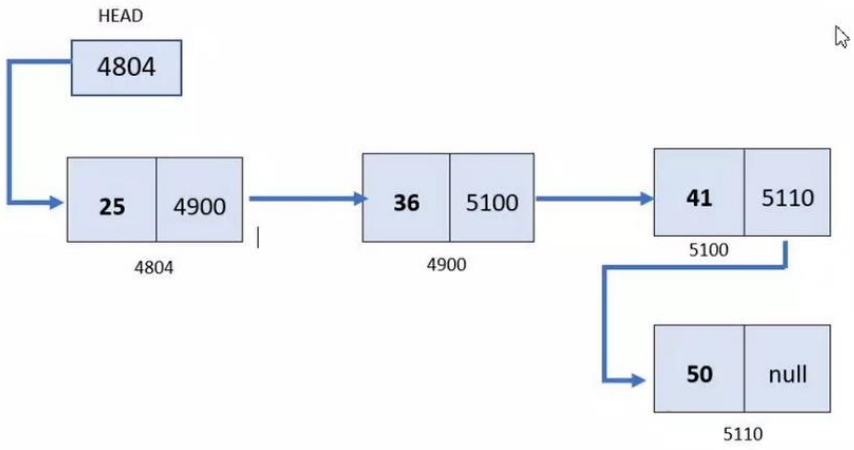
Inserting a new node in between two nodes.

1. Point new NewNode.next to RightNode.
2. Then the next node at the left should point to the new node.
3. This will put the new node in the middle of the two.



Exercise:

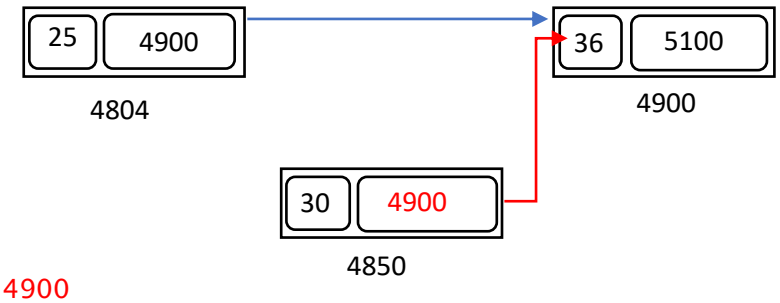
1. Consider the following linked list and answer the following questions.



a) Draw the graphical representation for the following options.
b) Insert Node 4850 in between node 4804 and Node 4900.
c) Insert Node 5115 to the end of the linked list.
d) Delete Node 4804.

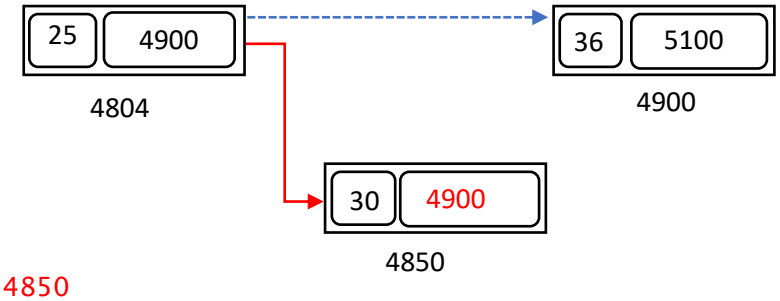
b)

step 1:



- 4850.next → 4900

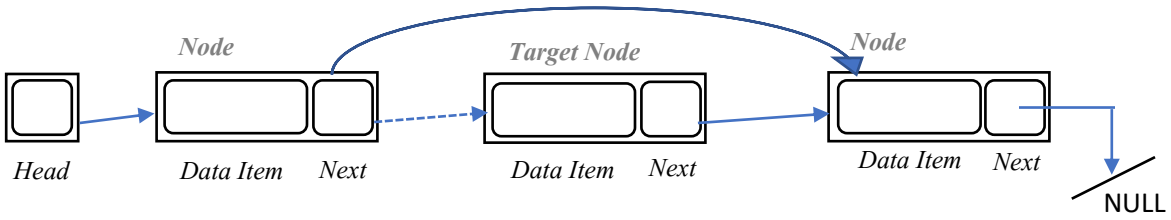
step 2:



- 4804.next → 4850

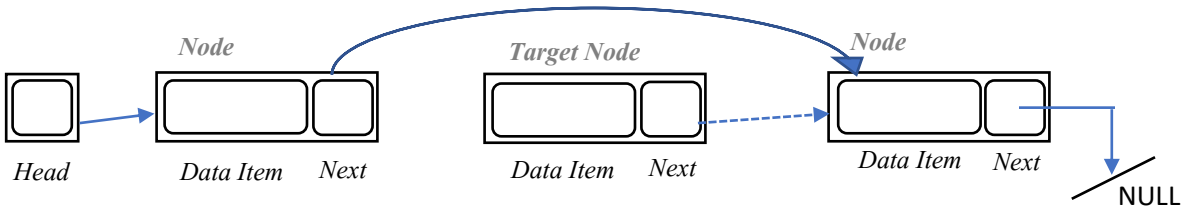
Deletion

First, locate the target node to be removed, by using searching algorithms.

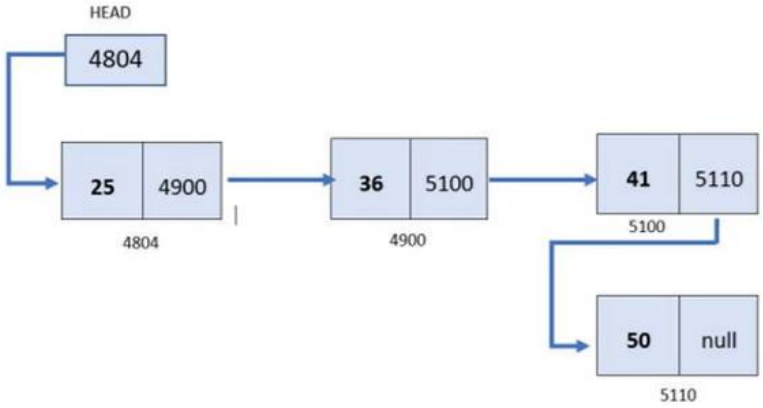


This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.

Target.Node.next → Null

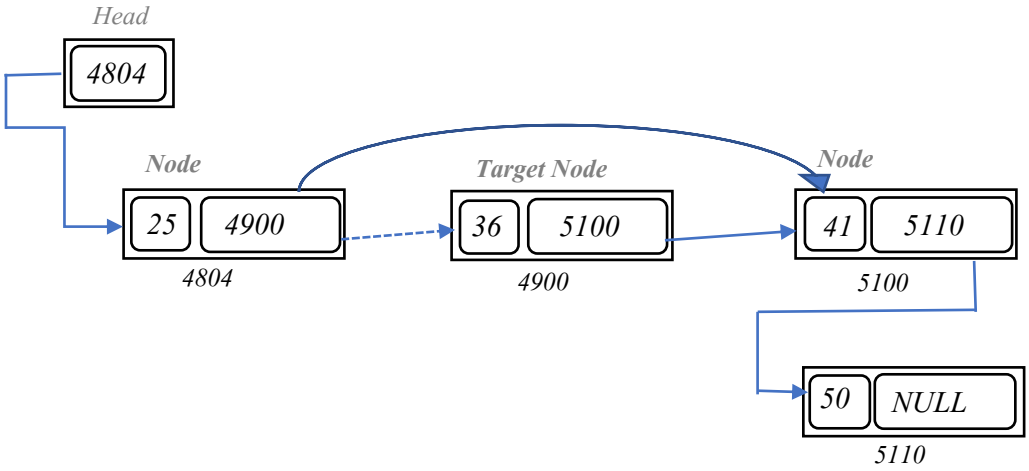


Exercise:



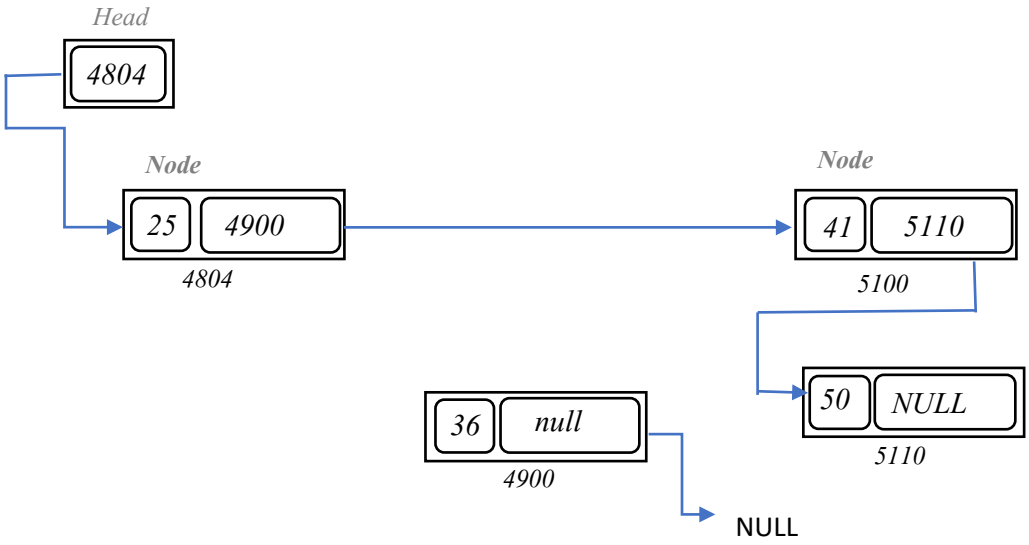
1. Delete Node 4900
2. Delete Node 5110

Step 1:



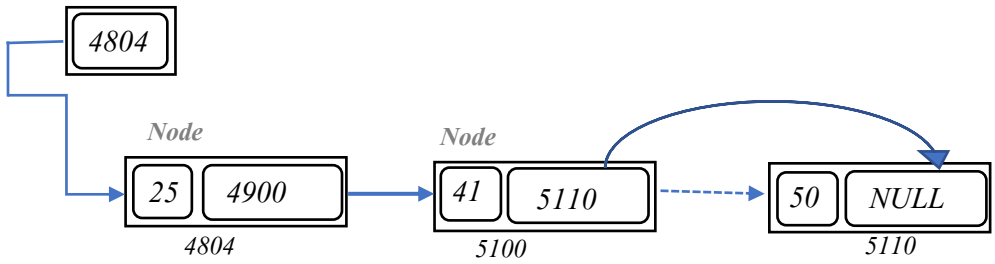
♥ 4804.next → 5100

Step 2:



♥ 4900.next → NULL

Q2]



Why linked list?

Arrays can be used to store linear data of similar types, but arrays have the following limitations.

- The size of the array is fixed.
- Inserting a new element in an array of elements is expensive.

Difference between arrays and linked lists

Arrays	Linked lists
Have a pre-determined fixed size	No fixed size
Easy access to any element a[i] in constant time.	No easy access to the i-th element should start with the head of the list
No space overhead	Space overhead
	Each element must store an additional reference.
Arrays are not dynamic, which means you can't change it	Linked list is dynamic.
Number of nodes is fixed.	Number of nodes in a linked list is not fixed.

Introduction to recursion and recursive algorithms

- A recursive function is a function that calls itself.
- Recursive functions can be useful in solving problems that can be broken down into smaller or simpler subproblems of the same type.
- A base case should eventually be reached, at which time the breaking down (recursion) will stop.

Recursive functions

Consider a function for solving the count-down problem from some number `num` down to 0:

- The base case is when `num` is already 0: the problem is solved and we "blast off!"
- If num is greater than 0, we count off num and then recursively count down from `num`-1

Recursive functions: base case/ recursive call

```
void countDown(int num)
{
   if(num == 0)
      printf("Done printing\n");
   else
   {
      printf("%d\n",num);
      countDown(num-1); //recursive call
   }
}
int main()
{
   countDown(10);
   return 0;
}
```

There are two types of recursions.

1. Direct recursion → a function calls itself
2. Indirect recursion → function A calls function B, and function B calls function A
   Or, function A calls function B, which calls…., which calls function A

1. Write a program in C to find the factorial of a number using recursion.

```
#include <stdio.h>
int facto(int x)
{
   if(x==0)
      return 1;
   else
      return x*facto(x-1);
}
int main()
{
   printf("fact is %d\n",facto(5));
   return 0;
}
```

2. Write a program in C to print even or odd numbers in given range using recursion.

```
#include <stdio.h>
void even(int first,int last)
{
   if(first>last)
      printf("done\n");
   else{
      if(first%2==0)
      {
         printf("%d\n",first);
         even(first+2,last);
      }
      else
      {
         first=first+1;
         printf("%d\n",first);
         even(first+2,last);
      }
   }
}
int main()
{
   even(2,12);
}
```

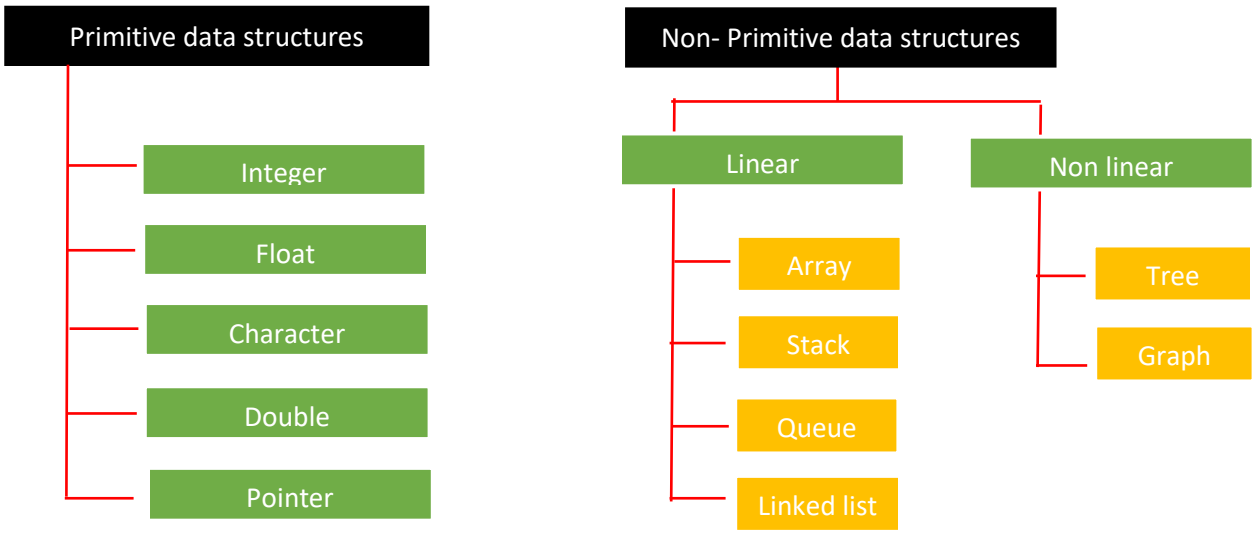
3. Write a program in C for binary search using recursion.

```
#include <stdio.h>
int bsearch(int a[], int start, int end, int key)
{
   if(end >= start)
   {
      int middle = start+(end - start)/2;

      if(a[middle] == key)
         return middle;
      if(a[middle] > key)
         return bsearch(a, start, middle-1, key);
      return bsearch(a,middle+1,end, key);
   }
   return -1;
}
int main(void)
{
   int a[]={1,4,7,9,16,56,70};
   int n = 7;
   int key = 9;
   int foundkey = bsearch(a,0,n-1,key);

   if(foundkey == -1)
   {
      printf("element is not found in the array");
   }
   else
   {
      printf("element found at %d",foundkey);
   }
   return 0;

}
```

08/02/2022lec12

Trees

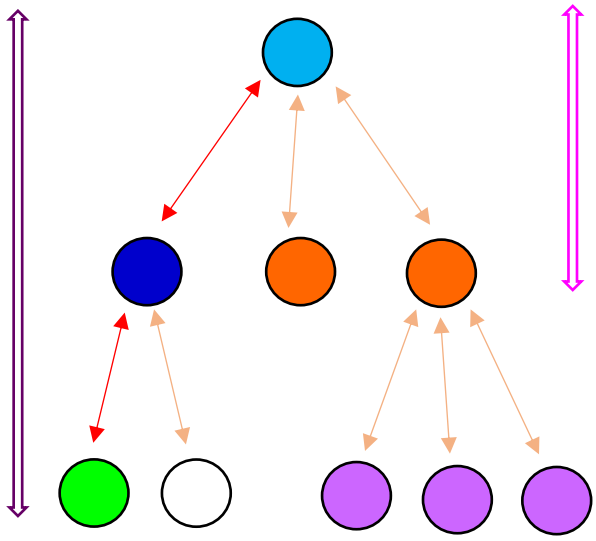
Data structures



Tree data structure

- Extension of linked list structure
  - → Each node connects to multiple nodes
  - → Example usages include file systems, Java class hierarchies

Tree terminology

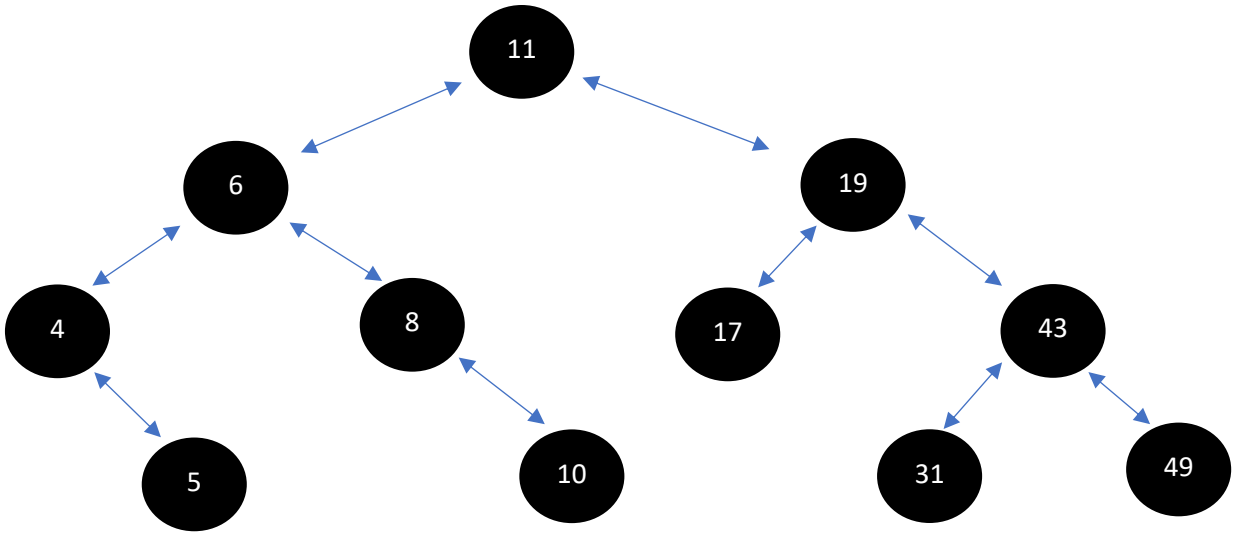


- Just like linked lists, trees are collections of nodes.
- Conceptualize trees upside down (like family trees)
- The top node is the **root**.
- Nodes are connected by **edges**.
- Edges define **parent** and **child** nodes.
- Nodes with no children are called **leaves**.
- Nodes that share the same parent are **siblings.**
- A **path** is a sequence of nodes such that the next node in the sequence is a child of the previous.
- A node's **depth** is the length of the path from root.
- The **height** of a tree is the maximum depth.
- If a path exists between two nodes, one is an ancestor, and the other is a descendant.

Binary search tree (BST)

- BST can't have more than two children.
- All the less nodes are in the left side & all the greater nodes are in the right side.

Draw a binary search tree by inserting: 11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31



15/02/2022lec13

Pre order: 16, 8, 4, 5, 6, 20, 18, 19, 25, 30

Post order: 6, 5, 4, 8, 19, 18, 30, 25, 20, 16

In order: 4, 5, 6, 8, 16, 18, 19, 20, 25, 30

Bread First Search & Depth First Search

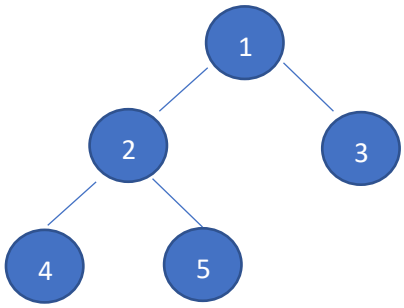
Bread first search = exploring the same level wider

Depth first search = exploring the depth further

Searching nodes

- Exploring a vertex (DFS)
- Visiting a vertex (BFS)

Demonstration



BDF: 1,2,3,4,5

DFS: 1,2,4,5,3

DFS: A, D, C, E, B

BFS: A, B, D, C, E / A, D, B, C, E

BFS: 1,2,4,3,5,7,8,6,9,10

DFS: 1,2,7,8,5,6, 3,10,9,4

22/02/2022lec14