22/07/24

Datastructures

- Organized collection of data along with Permissible operations
- Data structures is mainly classified based on Arrangements.
  - linear datastructures.
  - Non-linear Datastructures
- Types of Data - Homogenous
  - Non Homogenous / Hetrogenous.
- size - static Datastructure
  - Dynamic Data structure.

23/07/24

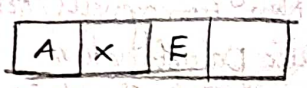
- The main datastructures are Array, stack, queue, Tree and graphs
- Arrays → linear, static, Homogenous
- stack → linear, Dynamic, Hetrogenous.
- Queue → linear, Dynamic, Hetrogenous
- Tree, Graph → Non linear, Dynamic, Hetrogenous
- Linked list → linear, Dynamic, Hetrogenous

- Arrays - elements are Arranged in a linear fashion
  - The elements are Accessed w.r.t the location (index)
  - operations - Insertion
    - Deletion
    - Search
    - Merge / split
    - Sort.
- In an Array if we known the Address of first value so we can Access the next Memory cells since it is Contigous.

- The increment of contigous memory cells will be regarding to the data type of member stored.
- The Index starts from zero in the Implementation Pgm[0]
- Array is access is Random since we don't need to worry about the Memory location since it is index based and contigous.

Insertion in Array

- Insert to the end



Occupied index = 1 Arr[OI + 1] = value

Arr[6] = Value./E

AMXE → A[3] = E

A[2] = M

A[1] = X

24-07-24

- Insertion at a particular position (Occupied/Vaccant cases)

Insertion at a position(arr, pos, val)

```
if arr[pos] = -1
   then arr[pos] = val      } If pos is vaccant
for (i = size-1 to pos)     } Decrement loop from the Rightmost
   arr[i] = arr[i-1]          position
   arr[pos] = val
if (pos ≥ size || pos < 0)  } invalid position
      return -1
Count = 0
   for (i = pos to size)    } Array full/not.
      if (arr[i] = -1)
      Count = Count + 1
```

Search in Array

Array → Sorted / unsorted

Output of search → Successful / unsuccessful

Basic operation in Array for search is comparison.

Linear Search Algorithm

```
Linear Search(Arr, ele, Size)
if ele = Arr[i]
return i
```

```
for (i = 0 to size)
return -1
if Ele = arr[i]
return i
```

Q Apply linear Search on the given and identify the number of comparisons performed for each given value

72 02 52 14 17 86 23 39

i) 72

72 = arr[0] = 72 ✓

1 comparison

ii) 39

39 = arr[0] = 72 ×

39 = arr[1] = 02 ×

39 = arr[2] = 52 ×

39 = arr[3] = 14 ×

39 = arr[4] = 17 ×

39 = arr[5] = 86 ×

39 = arr[6] = 23 ×

39 = arr[7] = 39 ✓

8 comparison

iii) 17

17 = arr[0] = 72 ×

17 = arr[1] = 02 ×

17 = arr[2] = 52 ×

17 = arr[3] = 14 ×

17 = arr[4] = 17 ✓

5 comparison

iv) 57

57 = arr[0] = 72 ×

57 = arr[1] = 02 ×

57 = arr[2]

57 = arr[3]

57 = arr[4]

57 = arr[5]

57 = arr[6]

57 = arr[7]

8 comparison ele not found.

unsuccessful search.

Q. Program to find the maximum element in an Array
Q. find the minimum element in an Array
Q. display all values in odd positions
Q. display all values in even position
Q. Swap first and last values in character Array
Q. Swap first and last values in an Integer without a temp Variable

```
#include <stdio.h>
Int main() { int arr[];
for (i=0; i<=5; i++)
{
Printf("Enter the array:");
for (i=0; i<=5; i++)
{
Scanf("%d", &arr[i]);
Printf("%d", arr[i]);
}
```

for (i=0; i<=5; i++) {

1.
```
#include <stdio.h>
int main()
{ int arr[5] = {23, 12, 45, 20, 90, 89};
 int max = arr[0];
 for(i=0; i<5; i++)
  { if(max< arr[i])
    max= arr[i];
  }
 Printf("Array elements:");
 for(i=0; i<5; i++)
  { Printf("%d", arr[i]);
  }
 Printf("The maximum element in array is %d", max);
 return 0;
}
```

2.
```
#include <stdio.h>
int main()
{
 int arr[5] = {23, 12, 45, 20, 90, 89};
 int min = arr[0];
 for (i=0; i<5; i++)
  { if (main > arr[i])
     main = arr[i];
    }
 Printf("The Array elements", arr[i]);
 for (i= 0; i<5; i++)
 {
 Printf("%d", arr[i]);
 }
```

```
Printf("%d The Minimum element is %d", min);
return 0;
}
```

3.
```
#include <stdio.h>
int main()
{
 int arr[5];
 Printf(" The Array element is:");
 for(i=0;i<=5;i++)
  {
   Printf(" %d", arr[i]);
  }
Printf("The Array in odd positions are:");
for (i=0; i<5; i=i+2)
 {
  printf("%d\n", arr[i]);
 }
return 0;
}
```

4.
```
#include <stdio.h>
int main()
{
 int arr[5];
 Printf("The elements of Array");
 for(i=0; i<5; i++)
  {
   Printf("%d", arr[i]);
  }
Printf("The Array in even positions are:");
for (i=1; i<=5; i=i+2)
{ Printf("%d\n", arr[i]); } return 0; }
```

5.
```
#include <stdio.h>
int main()
{
 int arr[5];
 int x = arr[0], y = arr[5]; temp=x;
 Printf("The elements of Array are:");
 for(i=0;i<=5;i++)
 {
  Printf("%d", arr[i]);
 }
 int temp = x;
    x=y;
    y=temp;
 Printf("The Array after Swapping is");
 for(i=0;i<=5;i++)
 {
  Printf("%d", arr[i]);
 }
 return 0;
 }
```

The complexities of time and space categorized into many.

- Time complexity is constant complexity if the number of search comparison done is constant. These are called best case scenario.
- Analysis is done with respect to — best
  — worst
  — Average.
- linear search — best case

  Scenario
  element to be searched is in the 1st position
  Number of comparison independent of input size
  number of comparison always

  complexity is constant complexity
  O(1) — best case complexity
  O(n) — worst case
- units are used to denote complexity (Asymptotic Notation)
  - Big oh — O
  - Big omega — Ω
  - Theta — Θ
  - Small omega — ω
  - Small oh — o
  - Tilde — ~
- The scenarios of worst case in linear seach are element is found in the last position of array and the element is not found The number of comparison = number of inputs. So the comparison is linear.

Q. Apply the Appropriate on the given set of elements and search for 83 also identify the number of comparisons performed.

10	7	92	83	57

Q. write a search program to search a particular name in a given ranklist.

Binary search

- we use this method in sorted Array.
- Mid = (low + high) / 2
- mid == VAL ?
- if (mid < VAL)
- if true then go the ~~left~~ right part or else go to the left part
- BinarySearch (Arr, key, low, high)

```
if (low <= high) {
mid = (low + high)/2
if (Key = Arr [mid])
return mid
else if (Key < Arr [mid])
BinarySearch (Arr, key, low, mid-1)
else
BinarySearch (Arr key, mid+1, high)
}
else
return -1;
```

30-07-24

1.

```c
#include <stdio.h>
int Maximum(int arr[],int n)
 {
  int max=arr[0];
 for(int i=1;i<n;i++)
  {
   if(arr[i]>max)
    {
     max=arr[i];
    }
  }
 return max;
 }

int main()
 {
 int arr[]={1,2,3,4,57,13}
 int n=Size of(arr)/Size of(arr[0]);
 int max= Maximum(arr,n);
 Printf("The maximum element in the Array is: %d",max);
 return 0;
 }
```

2.

```c
#include <stdio.h>
int Mini(int arr[], int n)
  {
    int min = arr[0];
   for(int i=1; i<n ; i++)
    {
     if (arr[i] < min)
       {
         min = arr[i];
       }
    }
  return min ; }

int main()
 {
 int arr[]={98,81,100,20,10};
 int n = Size of(arr)/ Size of(arr[0]);
 int Min = Mini(arr,n);
 Printf(" Minimum element in the Array is %d", min);
 return 0;
 }
```

3

```c
void OddPos(int arr[],int n)
{
 Printf("Elements in the odd position are:");
 for(i=0; i<n ; i = i+2)
  {
   Printf("%d", arr[i]);
  }Printf("%d\n");
}
int main()
  {
  int arr[]={1, 2, 3, 99, 34};
  int n= Size of (arr)/ Size of (arr[0]);
  OddPos (arr,n);
  return 0;
  }
```

4.

```c
Void EvenPos (int arr[],int n)
 {
  Printf("Elements in the even Position are:");
  for (i=1; i<n ; i = i+2)
   {
    Printf("%d", arr[i]);
   }
 }
```

```
int main()
{
 int arr[] = {29, 81, 35, 2, 5, 7};
 int n = sizeof(arr)/sizeof(arr[0]);
 EvenPos(arr, n);
 return 0;
}
```

5.

```
#include <stdio.h>
Void swapFirstLast(int arr[], int n)
{
 if(n>1)
 {
  int temp = arr[0];
  arr[0] = arr[n-1];
  arr[n-1] = temp;
 }
}
int main() {
 int arr[] = {1, 2, 3, 4, 5, 99};
 int n = sizeof(arr)/sizeof(arr[0]);
 Printf("orgunal array");
 for(int i=0; i<n; i++) {
  Printf("%d", arr[i]);
 }
 swapFirstLast(arr, n);
 Printf("Afterswap");
 for(i=0; i<n; i++)
 {
  Printf("%d", arr[i]);
 }
 return 0;
}
```

6.

```
#include <stdio.h>
Void swapFirstLast(int arr[], int n)
{
 if(n>1) {
 arr[0] = arr[0] + arr[n-1];
 arr[n-1] = arr[0] - arr[n-1];
 arr[0] = arr[0] - arr[n-1];
 }
}
int main() {
 int arr[] = {1, 2, 3, 5, 99, 81};
 int n = sizeof(arr)/sizeof(arr[0]);
 Printf("orgfarray");
 for(i=0; i<n; i++)
 {
  Printf("%d", arr[i]);
 }
 SwapFirstLast(arr, n);
 Printf("The Array after swap is:");
 for(i=0; i<n; i++)
 {
  Printf("%d", arr[i]);
 }
 return 0;
}
```

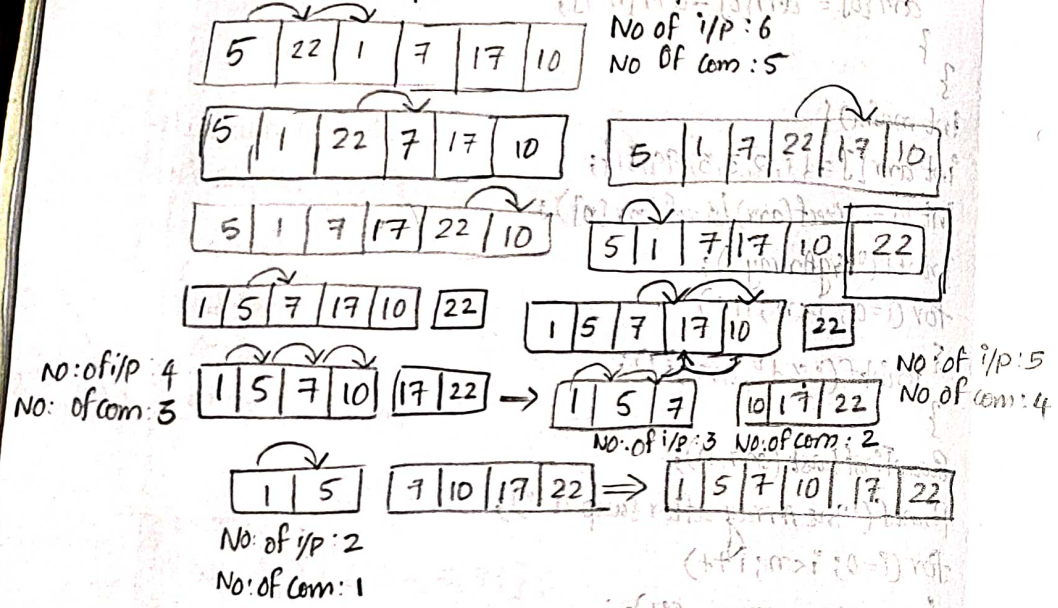
02-08-24 **Bubble Sort – inplace, adaptive**

It is the simplest sorting method.

Basically sorting algorithm will be in Ascending order Array.

Main operations performed are Comparison and Swap.

In the second comparison the last comparison in the second last.



Maximum number of comparisons $= \frac{n(n+1)}{2}$

Q. 5 4 3 2 1 → Bubble sort.

TEST PAPER-1

06-8-24

1. write the different data structure
   linear and non-linear
1.5 Homogenous and Heterogenous
+.5 structured and unstructured

2. Two features of Array:-
   - Random Access of elements through Index value.
   - Structured Arrangement of element of same data type under contigous Memory location

3. Replace third element of array.

```
temp = arr[2];
arr[2] = val
```

4. Array of 15 numbers. How many comparisons Require to Search for an element w.r.t linear Search in last position

$n(n+1) = 15(15+1) = 240$ comparisons

15x
16
90
15
240

5. Which Search is Appropriate for Search an element in an attendance Register w.r.t Register Number

Binary Search

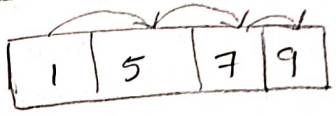
6. left sub Array Binary Search call Val < arr[mid]

7. what is the stopping condition in a Binary Search call? Recursive.

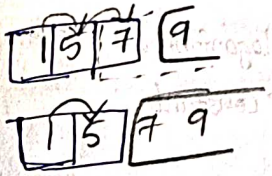
arr[mid] = Val

low > high

06-08-24



How many Pass Required to compare in Bubble Sort



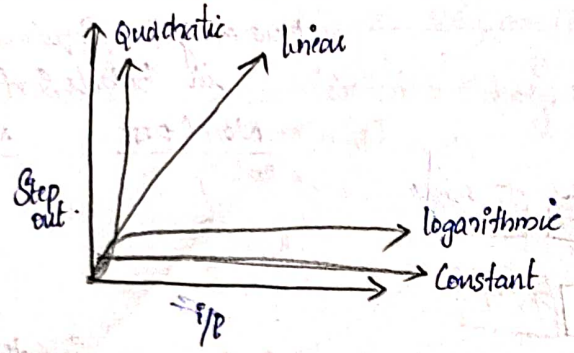
No: of pass	No: of comparison
1	3
2	2
3	1

If there is no swap in the first comparison then we can say that the Array is already Sorted.

```
Bubble Sort (Arr, n)
{ for (i=0; i<n-1; i++)  ---> Swapped = false;
  for (j=0; j<n-i-1; j++)
  {
    if arr[i] > arr[i+1] ;
      temp = arr[i] ;
      arr[i] = arr[i+1] ;
      arr[i+1] = temp;
      Swapped = true;
  }
  Swapped = False;
  break;
}
```

Best case : Already Sorted in Ascending order : n-1 comparisons
linear complexity since no: of elements & number of comparisons

Worst case : Descending order : Sum of first (n-1) natural numbers.
$\frac{n^2-n}{2}$ Quadratic Complexity



08/08/24 **Insertion sort** – inplace (NO Additional variable for Memory location for Sorting ~~storing~~)

```
insertionSort(arr,n)
{
 for(i=0; i<n; i++)
  {
   key = arr[i]
   j = i-1
   while((key < arr[j] && (j≥0)
    {
     arr[j+1] = arr[j];
     j--;
    }
   arr[j+1] = key;
  }
```

1. 15 10 9 6 – 3 pass

	Comp	shift
I st pass	1	1
II nd pass	2	2
III rd pass	3	3

worst case: Sorting in Descending order.

If number of input is 'n' and array is sorted in descending order.

Complexity ~~of~~ is n^2

2. 3 5 11 26 – I st pass – 1 comparson – shift 0
II nd pass – 1 – 0
III rd pass 1 – 0

Generally, if n element there is (n-1) comparisons. This is linear complexity. its the best case.

3. 2 7 11 36 – 27 29

	comparison	shift
I st pass	1	0
II nd pass	1	0
III rd pass	1	0
IV th pass	2	1
V th pass	2	1
	7	

– For partially sorted input ~~inp~~ insertion sort works fast.

Q which is the best Algorithm for partially sorted Array?

– If the number of elements is very small insertion sort is better.

Assignment:- Selection sort, Algo, working, example, complexity.
~~Shell~~

– 5 3 7 3 → 3 3 5 7

Bubble sort 5 3 7 3 – 3 5 7 3 – 3 5 3 7

3 3 5 7

Insertion sort 5 3 7 3 1st pass key = 3, comparison = 1, shift = 1

2nd pass key = 7, comparison = 1, shift = 0

3rd pass key = 3, comparison = 3, shift = 2

08/08/24

Lab Sheet-2

Refer: Prototype declaration
Void fnname( )
Col size Mandatory

Q write c programs using functions to

1. Find the maximum value in an Array.
2. Find the average of an Integer array.
3. To Replace an array with position (Accept array, Size of array, Position, new value)

Implement Search Algorithms.

4. Search an Integer in an array of integers
5. Search a character in your name
6. Search a name in array of names.
7. Search a number in a Sorted integer Array.
8. Search a character in Alphabets.
9. Search a number in an integer Array, if its found then swap its left and Right Values

9-08-2024

ASSIGNMENT-1

Selection Sort

Selection Sort is an in-place Sorting Algorithm Selection Sort works well for small files. it is used for Sorting the files. with very large values and small keys. This is because of the fact that selection is made ~~because~~ based on keys and swaps are made only when Required

Idea behind Selection Sort

The main idea is to Repeatedly select the smallest (or largest) element from the unsorted portion of the list and place it at the begining of the sorted portion. This process is Repeated until the entire list is sorted.

Selection Sort Algorithm

Step 1: Find the minimum value in list
Step 2: Swap it with the value in the current position
Step 3: Repeat this process for all the elements until the entire array is Sorted

This Algorithm is called Selection sort it Repeatedly selects the smallest element.

```
Void Selection (int A[], int n)
 { int i, j, min, temp;
  for (i=0; i<n-1; i++)
   {
    min = i;
   for (j=i+1; j<n; j++)
    { if (A[j] < A[min])
```

```
            min = j;
        }
        temp = A[min]
        A[min] = [AE i];
        A[i] = temp;
    }
}
```

Example of selection Sort

eg:- 64, 25, 12, 22, 11

1st pass :- Find the Smallest element 11 and swap with first element

Array after swap: 11, 25, 12, 22, 64

2nd pass – Find the Smallest element in the unsorted part and swap

Array after swap: 11, 12, 25, 22, 64

3rd pass: Find Smallest element 22 and swap

After swap: 11, 12, 22, 25, 64

4th pass: Find Smallest element 25 already in place So No Swap

Sorted Array: 11, 12, 22, 25, 64.

Analysis of selection Sort

- Time Complexity – Best → $O(n^2)$
  - worst – $O(n^2)$

The time complexity is $O(n^2)$ in all cases because there are 2 nested loops one iterating over each element (n times) and another iteration to find the smallest element.

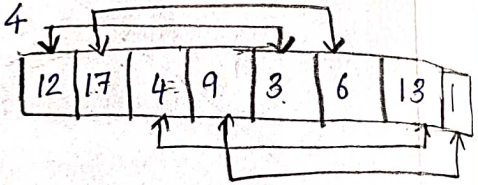
- Space Complexity :- Selection Sort is an inplace Sorting algorithm. which of requires only a constant amount of additional Memory Space.

Best case :- The best case Scenario also takes $O(n^2)$ times as the Algorithm always iterates over the entire list to find the Smallest element, regardless of whether the list is already Sorted.

Worst case :- The worst case Scenario occur when the list is Sorted in reverse order, but since the Algorithm always performs the Same number of comparisons and swaps, the time complexity remains $O(n^2)$

08/ 15-08-24 Shell Sort

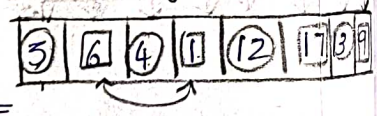
- Inventor Donald shell
- Generalization of insertion Sort
- Improves Insertion sort by comparing and exchanging elements that are far apart.
- Sort elements at specific intervals using Insertion sort
- i : interval
  N: size of array
  for i from N/2 to 1, with step i = i/2 do ($N/2, N/2/2, N/4/2 - \ldots$)
  Put all the elements which are at the distance of i in a sublist
  Sort all sublists using Insertion Sort.
- Arr[] = {12, 17, 4, 9, 3, 6, 13, 1}, N = 8.
  Sequence (N/2, N/4, ... 1) intervals
  Iteration 1 : Interval = N/2 = 4



  Sublist 1 : {12,3}
  2 : {17,6}
  3 : {4,13}
  4 : {9,1}

  Apply insertion Sort in all sublist element. Compare sublist ele- as key if it is lesser then insert and interchange

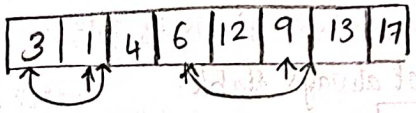
  3, 6, 4, 1, 12, 17, 13, 9



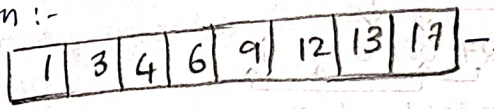
  Iteration 2 : Interval = N/4 = 8/4 = 2
  Sublist 1 : {3, 4, 12, 13} - already sorted.
  2 : {6, 1, 17, 9}

  After iteration 3, 1, 4, 6, 12, 9, 13, 17

Iteration 3 : Interval = N/8 = 8/8 = 1
every element is compared



After iteration :-



— Sorted Array.

Shell Sort - Algorithm

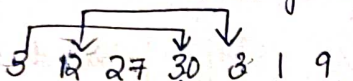
```
ShellSort (list, size)
{
  for (i=size/2 ; i>=1; i=i/2)
  {
    for (j=i ; j<size ; j++)
    {
      key = list[j];
      k=j;
      while (key < list[k-i] && k>=i)
      {
        list[k] = list[k-i]
        k = k-i;
      }
      list[k] = key;
    }
  }
}
```

Exercises

1. 9, 7, 1, 4, 3, 8, 6, 2, 10, 5
2. 12, 4, 3, 9, 18, 7, 2, 17, 13, 15, 6

14/08/2024 Shell Sort Features

- Adaptive – depends on how sorted the data is, depend on seq. chosen
- inplace
- stable – is not always stable.



iteration 1: Interval 7/2 = 3

Sublist: ~~{5,30}~~ {3, 30, 9} 3 3 1 9 12 27 30

{12, 3} ~~3 9 3~~

{27, 1}

~~{30, 9}~~ → {3, 12, 27, 30, 3, 1, 9}

7/4 = 1.

~~{3, 27, 3, 9}~~

~~{12, 30, 1}~~

{3, 3, 1, 9, 12, 27, 30}

{1, 3, 3, 9, 12, 27, 30}

{3, 27, 3, 9, 12, 30, 1}

{3, 3, 27, 9, 12, 30, 1}

{3, 3, 9, 27, 12, 30, 1}

{3, 3, 9, 12, 27, 30, 1}

{1, 3, 3, 9, 12, 27, 30}

Complexity

Best – O(n log n)

worst – O(n²) linear + arithmetic = linearithmitic

Average – O(n log n)

Shell sort Applications

- used in linux kernel
  linux kernel do not use stack to store call addresses.
- uClibc uses shell sort.
  a standard c library intended for linux-kernel based os for embedded systems.
- uses in bzip2 compressor with knuth's sequence.
  sort before compression – improve the compression ratio and reduce the size of compressed file
- cyptography
- Database indexing.
  SQLite – B tree for indexing – use shell sort to build and maintain B-Tree.

14-08-2024 ASSIGNMENT-2

Linear Search

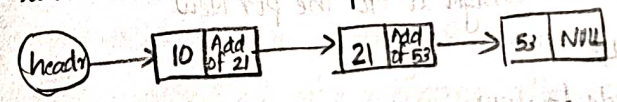
1. In an unsorted array, determine if there are any duplicate elements in the array?
2. Find if there is an element in an array that meets a specific which is being greater than ~~equal~~ a certain value
3. Find maximum value in an unsorted Array
4. Find the Repetition of certain value in an array.
5. Find if there are consecutive identical elements in an array

17/08/2024

Linked list

- linear data structure
- Dynamic
- Heterogeneous
- Series of connected nodes. The entity in the linked list is called node. logical connection exist between the nodes.

- The head is the pointer to the initial node (Address of 1st node)
- The last node's Address part will be null.



Abstract Data Types

* Every data structure has a Abstract datatypes which specifies the functionalities and operation that can be performed in the data structure. it is meant for the user.
* Datastructure is the idea of how the operations and implementation in detail then it is called a datastructure it is in developer Perspective.

Creation of Linked List

Create a node
if(head = NULL) (list empty?)
head = new (new node ptr new to head 1st node)
else (if it is not null)
Traverse upto the last node. (you traverse and add node to last pos)
Assign last node as temp
temp → next = new

Algorithm for linked list

1. with 2 pointers

First ptr → pointing to the first node (head)

Second ptr → pointing to the last node. (last)

- Start
- head = NULL
- Last = NULL
- Create a node and assign it to the ptr New
- if (head = NULL)
- Assign new node to NULL.
  Assign new node to last.
- else
  Assign next part of last pointer to new.
  Assign new node to last pointer

22/08/2024

Pointers

- Pointers are special variable used to store Addresses.
  datatype *variablename; (To access a value)
- data type denotes the type that can only hold by the pointers
- eg: int *n;
- '&' is used to Represent Address where * is used to access value in a Respective address space
- Before the execution memory is allocated ie, during Compile time
- Dynamic allocation ie, during Runtime, 3 methods used
  malloc - Specified size memory allocated
  Calloc - Memory is allocated and initialized with zero
  Realloc - Reallocates the memory allocated.
- malloc (sizeof(int)); - 4 bytes allocated.
  malloc (sizeof(2*int)); - 8 bytes allocated.
- Stack based Memory - Compile time Memory
  heap based Memory - Runtime Memory
  } Two types of Memory allocatio
- malloc returns the address of the first byte after allocation.

Structure

- User defined data Structure which is Heterogenous datastructure

```
Struct Student
 {
  int regno;
  char name[20];
 };
```

- We can create normal variable | Pointer Variable
- Structure object can be created in three ways:

a) Single object

Period operator can be used

```
Student s;
Stud.name; → Access the element's value.
S[i].regno;
```

b) Pointer object

```
struct student *s;
```

→ is used to access value of variables

def type
Create memory location and assign

LAB SHEET 3 – 22-08-2024

1. Bubble sort for Integer array
2. Selection Sort for character Array
3. Insertion Sort for Integer Array
4. Shell sort for Integer Array

```
#include <stdio.h>
void bubblesort(int arr, int n)
{
 int i, j, temp;
 for(i=0; i<n-1; i++){
 for (j=0; j< n-i-1; j++)
  { if (arr[j] > arr[j+1])
    { temp= arr[j];
      arr[j] = arr[j+1];
      arr[j+1] = temp;
    }
  }
}
}
void readarray (int arr[], int size)
 { int i;
   for(i=0; i< size; i++)
    Printf ("%d", arr[i]);
    Printf ("\n");
 }
 int main()
 {int arr[] = {64, 25, 12, 22, 11};
  int n = sizeof(arr)/ sizeof(arr[0]);
  Printf("Unsorted array: \n");
  Print readArray(arr, n);
  bubblesort(arr, n);
  Printf("Sorted Array: \n");
  Prin readArray(arr
```

24-08-2024 Insertion in Linked List

1. Insertion at the begining
2. Insertion at the end
3. Insertion after a node
4. Insertion ~~aft~~ before a node
5. Insertion of a node at a given position

1. Insertion at the begining

Create a node new
new → next = head
head = new.

```
if(head==null) then
    head = new
else
    head = new
```

Algorithm

```
Insert to the first (Struct node * head, int val)
{
  Struct node * new = create node(val)
  new → next = head ;
  head = new ;
  return head ;
}
```

2. Insertion at the end

Creat a new node.

```
while (temp != NULL)
  temp = temp → next
```

For traversal
The general Algorithm.

```
temp = head
while (temp → next != NULL)
  temp = temp → next
if (temp → next = NULL)
  temp = new
```

3. Insertion after a node

```
create a new node
temp = head
while (temp → data != 20)
temp = temp → next.
new → next = temp → next.
temp → next = new .
```

Algorithm

```
Insert After (Struct node * head, int val, int key)
{ Struct node * new = create new node (val);
  temp = head
  while (temp → next != NULL)
  while (temp → data != 20)
  temp = temp → next
  new → next = temp → next
  temp → next = new
}
```

4. Insert before a node

```
Insert before (Struct node * head, int val, int key)
{ Struct node * new = create node (val)
  temp = head
  while (temp = next → data != key)
     temp = temp → next
  new → next = temp → next
  temp → next = new
}
```