Page No.
Date

20/4/22

Q Write a prog to input a number and display.

```
#include <stdio.h>
int main()
{ int a = 10;
  printf("%d", a);
}
```

```
int main()
{ int a;
  Printf("Enter no");
  Scanf("%d", &a);
  printf("%d", a);
}
```

Enter 5 numbers

disad
- 5 variable

```
int main()
{ int num1, num2, num3, num4, num5;
  printf("Enter 5 numbers");
  scanf("%d %d %d %d %d", &num1, &num2
                          &num3, &num4, &num5);
  printf("You entered %d %d %d %d %d,
          num1, num2, num3, num4, num5);
  return 0;
}
```

Storage

adv
- 2 variable

disad
- data loss

```
int num;
printf("Enter number");
scanf("%d", &num);
printf("%d", num);
```

→ for(i=0; i<5; i++);

(1) all have same name ← Array → memory same ↑ → variable declared in one go (3)

→ read & write using loop

int main
{ int num[50];

(2) all variables in continuous block.

→ adv - arrays can work very fast. with less memory.

→ disadv - that much memory to be available.

* EFFICIENT HANDLE DATA

Disadvantage - work in static way.

(4) - need to give advance declaration
- Known how much data to be stored. 60 - 50 ✗ 60 - 150 ✗ eg:- Birthday
thats why - use dynamic alloc

static - compile time pe allocation
dynamic - work with prog. during execution / Run time.

Data structure - "Dynamically"

int arr[5]; ✓ ← - numeric constant
int arr[]; - ✗
int arr[] = {1, 2, 3};

pre processor - bco2 work before start prog.
Symbolic constant → # define
import constant # include

19/4/22

Design ⟶ Implementation.

	Design	Implementation
①	Algorithm (Seq of steps or inst. for achieving particular target.)	Program. (Implementation of algorithm)
	Steps & seq both are important.	
②	Done by domain specialist	② Done by coder
③	using English/NL/Maths.	③ using programming language.
④	No constraints	④ constrains H/W & acc / OS / APIs.

Algorithm of Take 2 nos & display sum.

step 1 Start

step 2 Read A, B

step 3 Display A + B

step 4 Stop

34
769

Page No.
Date

Algo for sum of n numbers

1. Start
2. Sum = 0
3. Read NUM
4. sum = sum + NUM
5. If more numbers available
   Goto step 3
6. Else
   Display sum
7. stop

PROPERTIES OF ALGORITHMS

1) Every algo must have input.
2) Every algo must display expected output.
3) Every algo must have finite no. of steps
4) Definiteness (Predeclared)
   ↳ every stmt must have meaning.
   other not be included in algo.
5) effectiveness - Algo effectiveness must be measurable.

```
# include <stdio.h>
int main()
{ int sum = 0, num, i, n;
  printf ("Enter no. of values to be terms");
  for (i=0; i<n; i++) scanf ("%d", &n);
  {printf ("Enter a number");
   scanf ("%d", &num);
   sum = sum + num; }
printf ("%d", sum);
}
```

20/4/22

§ Write a prog to input a number and display.

```
#include <stdio.h>
int main()
{ int a = 10;
  printf("%d", a);
}
```

```
int main()
{ int a;
  Printf("Enter no");
  Scanf("%d", &a);
  printf("%d", a);
}
```

Enter 5 numbers

disad
- 5 variable

```
int main()
{ int num1, num2, num3, num4, num5;
  printf("Enter 5 numbers");
  scanf("%d %d %d %d %d", &num1, &num2,
                          &num3, &num4, &num5);
  printf("You entered %d %d %d %d %d,
          num1, num2, num3, num4, num5);
  return 0;
}
```

storage

adv
- 2 variable

disad v
- data loss

```
int num;
printf("Enter number");
scanf("%d", &num);
printf("%d", num);
```

→ for(i=0; i<5; i++)

(1) all have same ~~name~~ ← Array → memory same ↑ → variable declared in one go (3)

Page No.
Date

→ read & write using loop

```
int main
{  int num[50];
```

(2) all variables in continuous block.

→ adv - arrays can work very
→ ~~diff mult~~ fast. with less memory.
→ disadv - that much memory to be available.

* EFFICIENT HANDLE DATA

Disadvantage - work in static way.

(4) - need to give advance declaration
- Knows how much data to be stored. 60 - 50 x 60 - 150 x eg:- Birthday

thats why - use dynamic alloc

static - compile time re allocation
dynamic - work with prog. during execution / Run time.

Data structures - "Dynamically"

int ~~[50]~~ arr[5]; ✓ ← - numeric constant
int arr[]; - x
int arr[] = {1, 2, 3};

preprocessor - bcoz work before start
Symbolic constant → # define & Prog.
import constant # include

LINEAR SEARCH

```
# include <stdio.h>
int main
{ int a,n,i, , arr[10]; for(i=0;i<10;i++
{ printf ("Enter array");
scanf ("%d", &arr[i]); }.

Printf ("enter element to be searched");
scanf ("%d", &n);
if (n == a[i])
{ for (i=0; i<10; i++)
    { printf ("element found at %d", i+1);
      break;
    } return 0;
} else printf ("Value not found");
```

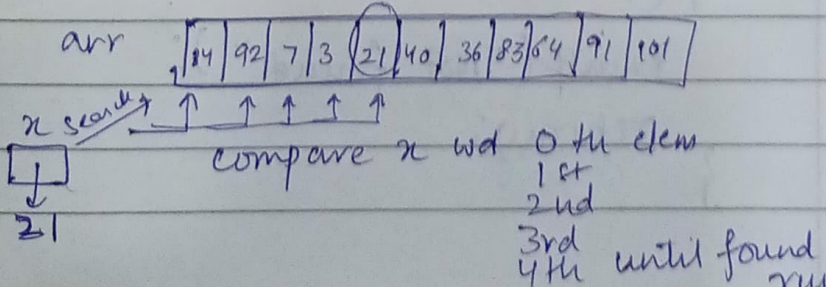
Algo to find a value in an Array of Numbers

step 1: Create an array.

step 2: Compare search value with all elements of given array using recursive / iterative method.

step 3: If value is found then display message and exit from program.

Step 4: If value not found then display an appt. message and exit.



Page No.
Date

No. milega ya nahi would be
done by (N) no. of operations.
(n)

PSEUDO CODE

Algo → to be written

given: ARR is an array of integers
and x is search value.

step1: FOR i in 1 to n LOOP
If x == ARR[i] then display
"value found" Exit
END LOOP.

step2: If i=11 then display "value not
found."

step3: EXIT.

CODE

```
int main()
{ int arr[10] = {34, 55, 66, 77, 22, 11, 99,
                 12, 19, 58};
  int x=99;
  int i;
  for (i=0; i<10; i++)
  { if (x == arr[i])
    {printf ("found");
     break;
    }
  }
  if (i==10)
  Printf ("Not found");
  return 0;
}
```

Page No.
Date

8 Write a function to perform linear search on given array.

```
void
~~int~~ search (int arr[10], int y)
  { int i;
    for (i=0; i<10; i++)
     { if (x == arr[i])
        { printf("found");
          break;
        }
     }
    if (i==10)
      printf ("not found");
  }
int main ()
  { int arr[10] = {36, 55, 66, 77, 88, 99, 109,
                   45, 100, 84};
    int x;
    printf ("enter num to be searched");
    scanf ("%d", &x);
    linear_search (arr, x);
    return 0;
  }
```

```
int main()
{ int arr[5] = {1,2,3,4,5};
  int i;
  int x = 3;
  for (i=0; i<4; i++)
   { if (x == arr[i])
     {printf("found")
      break;
     }
   }
  if (i==5)
    Printf("Not found")
  return 0;
}
```

step-1 ~~Create and array.~~

given: An arr and x value to be searched.

step 1: Compare x with every element of array. one by one.

step 2: if found ~~prt~~ display element found

Linear search — WORST APPROACH as we need to search n no. of times.

Best case — x compared to 0th value & got. = 1 step

~~Worst case~~ Technically O(1)

Big O notations

worst case.

Linear se
Case

→ Extremel

arr 0 3

x 31

→ when

→ perfor
st

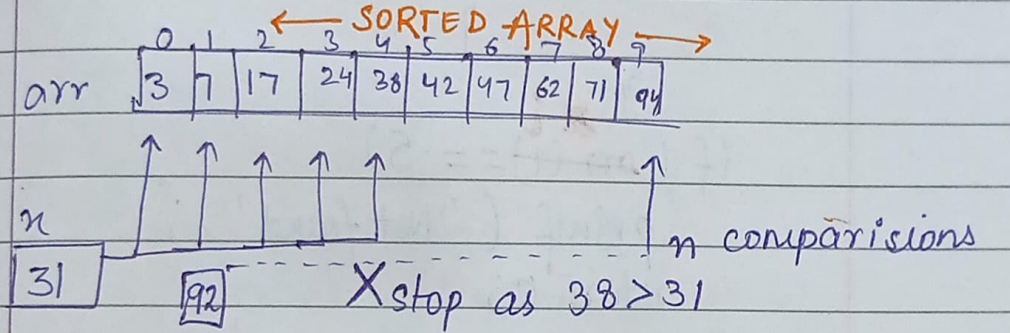
but
(n)

worst case → O(n) → n no. of operations at last step.

Linear search used in unsorted array

[we don't know where the search will be]

→ Extremely slow



→ when array is sorted.

→ performance improve if data found in starting.

but worst case scenerio still remains (n)

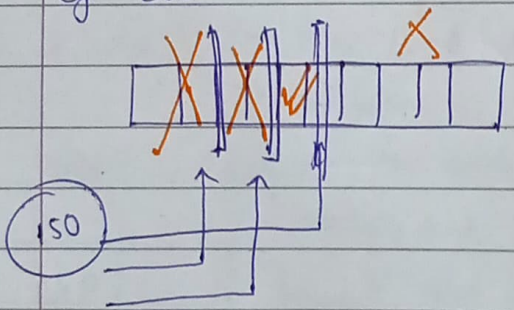
28/4/22

```
int main()
{ int arr[] = {1,2,3,4,5}
  int x = 3;  int i;
  for (i=0; i<4; i++)
    { if (arr[i] == x)
        { printf ("found");
          break;
        }
    }
  if (arr[i] == 5)
    printf ("Not found");
  return 0;
}
```

BINARY SEARCH

→ divide and conquer

→ eg: Book

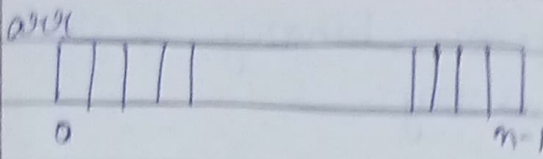


→ Total comparisions for 1000 data is 10

1000
500
250
125
65
32
16
8
4
2

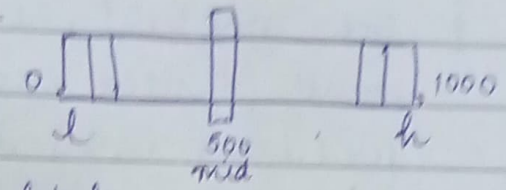
Comparision $\log_2(n)$

Q advantages of binary over linear?



n[ ]

low = 0
high = n-1
mid



① mid = $\frac{l+h}{2}$

② while (low < high)
{
if (x == arr[mid])
printf ("value found");
else if (x < arr[mid])
high = mid - 1;
else
low = mid + 1;
}
if (low == high)
Printf ("Not found");

Page No.
Date

Algo for Binary search.

Given: arr as a set of integers
x as search value
n as size of arr.

Start

step 1 low = 0 high = n-1

step 2 ~~to~~
while (low < high)
loop

step 3 Mid = (low + high) / 2

step 4 if x == arr (mid), then display "found" return / break;
else if x < arr [mid], then
high = mid - 1 ;
else low = mid + 1 ;
End If

step 5 End loop

step 6 Display "Not found" if (low > high)

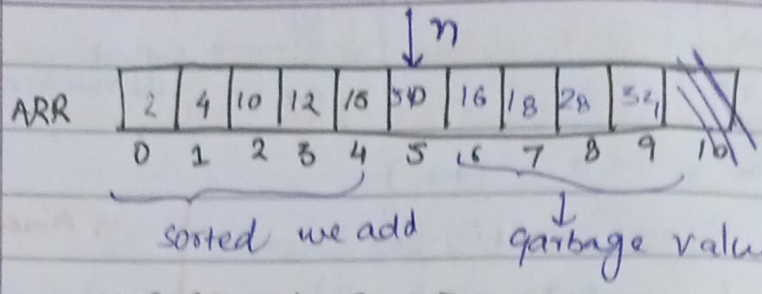
Stop

use

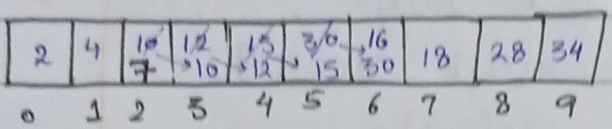
return - exit
↓
out of program
stop

break
↓
out of loop
direct step 6

2/5/22



INSERTION SORT



K → 7 → insert in order in array ARR

→ It will be between 4 and 10

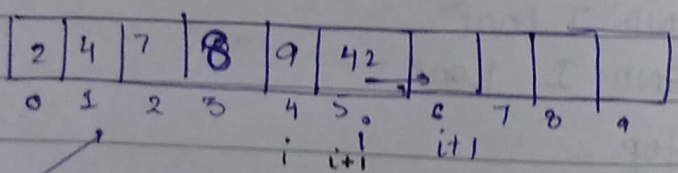
→ Before inserting shift every value by one.

1st shift - 30 2nd shift - 15

3rd shift - 12 4th shift - 10

then insert 7

- INSERTING ELEMENT IN SORTED ARRAY
- DELETING ELEMENT IN SORTED ARRAY



K [6]

arr[i+1] = arr[i];

i--

(4 pe aa jayega)

till

if (arr[i] > 6)

{

~~arr[i+1] = K~~ arr[i+1] = K

No need to compare with every element.

Worst case — work as bubble & selection sort.

average case — Insertion is better

ALGO

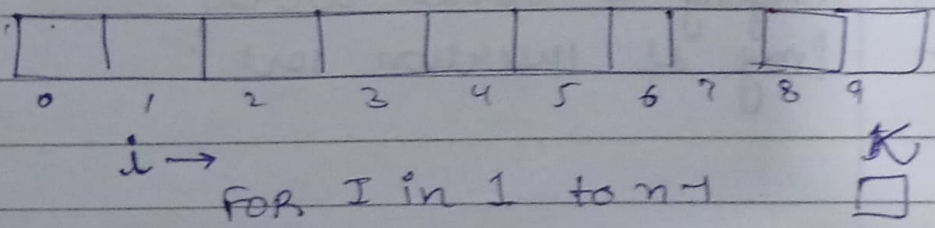
Step 1 Given arr as set of integers
K element to be inserted.

Step 2 FOR i from 1 to n-1

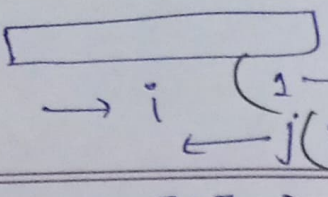
Step 3 IF (arr [i] > K)
then arr[i+1] = arr [i]
also ~~At~~ arr [i+1] = K
i--

Step 4 ~~stop~~ END IF
END I LOOP

Step 5 stop



FOR I in 1 to n-1
Save ith element to one place K
K = ARR [i];
Now ~~compare~~ start J loop from i-1 to 0
FOR J in I-1 to
Then compare jth element with K
& if greater then shift



if arr[j] > k then
arr[j+1] = arr[j]

Same condition till arr[j] > k

⇒ End loop when arr[j] > k then arr[j+1] = k.
END LOOP

PSEUDO CODE

```
FOR I in 1 to N
K = ARR[I];
J = I - 1;
while 'ARR[J] > k
    IF ARR[J] > k then
        ARR[J+1] = ARR[J]
END LOOP
ARR[J+1] = K
END LOOP
```

→ Merging 2 sorted array
→ Prog of insertion sort