~~Ⓢ ||| 2. Software Project vs Other Projects.~~

Data Structures & Algorithms

* Array.

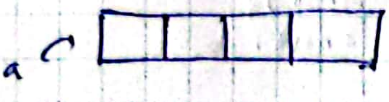
basic structure ⟶ variable.

arrays → control a number of values

student marks

int a 250
int b = 65
int c = 35
int d = 71
int e = 21

} 100 variables



contiguous

int ⊗ a[ ]

① index system

a[index] × base address

+[index] × [size of data type]

② pointer system

a^* + [size of data type]

contiguous = one after the other

* Multidimensional Array

→ a) array of arrays (array inside array).

~~int a~~ int a [3] [5]

↑ rows ↑ column

Arrays.

Pros

(complexity analysis)

- Fast access any element in a constant time.
- Homogenous → only one data type can hold
- Contiguous - one after the other

Cons

- dynamic resizability issue
  - ↳ [illegible] array, [illegible] value [illegible] add [illegible] [illegible] array [illegible] new [illegible] - looping

Exercise :- How to find size of a array

$$\frac{\text{size of (a)}}{\text{size of (0) index}} = \text{size of arr}$$

Exercise 02 :- How do this in multi dimensional array.

There are two ways to store data in memory by chipset

- ↳ row major order - eg: intel
- ↳ colum major order

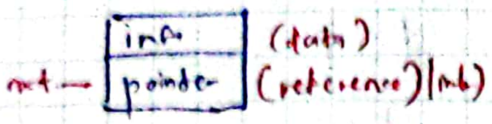
Linked List (in C - struct)

↳ why

↳ Arrays dynamic resizeability problem
↳ solution - dynamic list

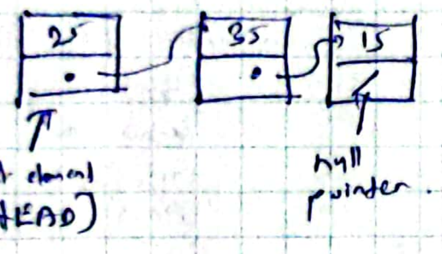
There are two ways that linked list works.

① Node Structure




```
struct {
    int a;
    int* ptr*;
}         (next)
```

Eg: ~~A[25, 35, 15]~~ A[] = {25, 35, 15};



Head - reference to 1st block
Head.info = 25
Head.next.info = 35
Head.next.next.info = 15

* Singly Lk. (SLL)

→ One link between node
→ □ → □ → □
→ HEAD → first element of the list
→ TAIL → last element of the list

* Insert Operations

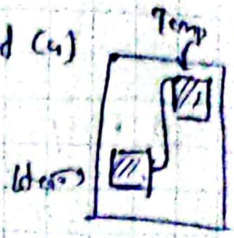
∘ putting new elements to a SLL
  • Where are we going to put?

① Befor HEAD
② After TAIL
③ In the middle
④ How do add to an empty list

• Data type matches?

• How do add to an empty list

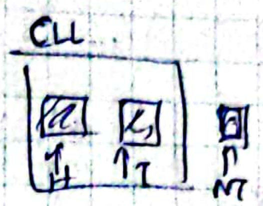
1) Head = Tail = ∅
2) Create a node element (a)
3) Head = Tail = a



• Add to Head

```
Temp.next = Head;
Head = Temp;
```

• Add to Tail

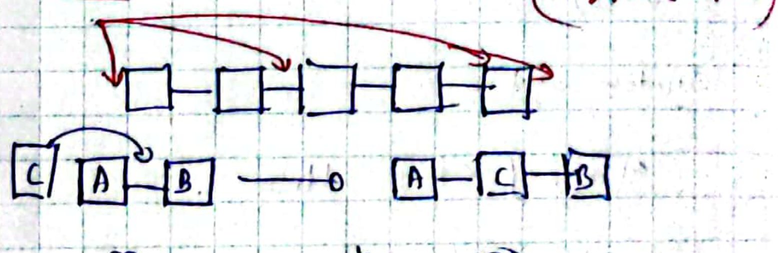



```
New Tail
~~NewTail.next = Tail;~~
Tail.next = newtail;
newtail = tail;
```

• Head = Tail = null

```
if (Head = null) {
    tail.head = next node
}
```

Add. (Insert Part)

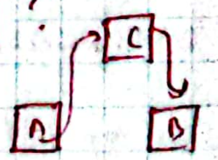


① temp1 = A
temp2 = B ✓

temp1.next = C
c.next = temp2

② A.next = C
C.next = B ✗

Why?



when we do according to 2nd way ~~there~~ A.next will work. when we doing that we remove a.next = B's reference. So c.next = B will not work because already we lost the list that's why!

(Finding Part)

Strategy 01 Counter (manual indexing)
↳ first set ~~count and over~~ to the linked list

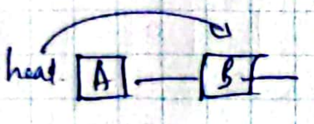
```
temp = head.
while (temp != 0; counter = 2){
    temp = temp.next;
    counter ++;
}
```

Strategy 02 Searching for a element

Delete

↳ ① Delete from head
↳ ② Delete from Tail
↳ ③ Delete from anywhere

① Delete from head / tail.



head = head.next. } head

① counting method.

```
temp.next = tail check.
~~temp~~ .
if (temp.next == tail){
  ~~temp.next = 1~~
  temp = tail
}
```

tail

② temp.next = null ✱
temp = tail.

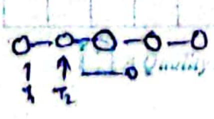
Qn.
Read from Tail only?

③ Delete from anywhere

A.next = B.next.

✱ if (temp.next ?)
t.next = t.next.next

✱ 2 pointer method
set t1 and the t.next t2.next.next)



Doubly Linked List

~~What is a data structure?~~
~~↳ A data structure is~~

Q1) Searching for a specific value in an array?
↳ calling to array by it's indexes.
Ex:- int arr[3] = {50, 60, 70}
printf("%d", arr[0]);

Q2) Testing for uniqueness and deleting duplicates in an array?
↳ ~~check if that~~ get first value of the array and check if there any duplicates in that array. If there any duplicates then that array index equals to zero for deleting the value. (2 for loops)

Q3) Finding the min/max in an array?
↳ store data of the first index of the array in to a variable. Then compare that variable with other indexes, ~~where~~ that is minimum or maximum. if that is minimum then if the min function new value ~~that~~ stored to the above variable.

```
int arr[3] = {50, 60, 70};
int min = arr[0];
int max = arr[0];

for (int i = 1; i <= 3; i++){
    if (min > arr[i])
      min = arr[i];
}

for (int i = 1; i <= 3; i++){
    if (max < arr[i])
      max = arr[i];
}
```

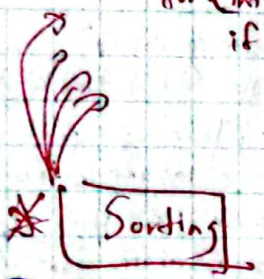
Q4) Counting how many time a specific value appear in an array?
↳ get the value from the user that need to check. Then compare that value with first array index whether it is equal or not.

if this equal then counter will increase.

Q) Set intersection/union between array 1 and another sorted array 2?
↳ ?

Q) Counting how many values in an array is inside range.
↳ define array, using first value of the first index into variable.
Then compare it with the given range. If it is in the range then count will increase.

```
↳ int arr[4] = {43, 45, 23, 32, 56};
  int a = arr[0];
  int count = 0;
  for(int i = 0; i < 5; i++){
      if((range1) < a < (range2))
          count = count + 1;
          arr[0] = arr[i];
```



Sorting Algorithms

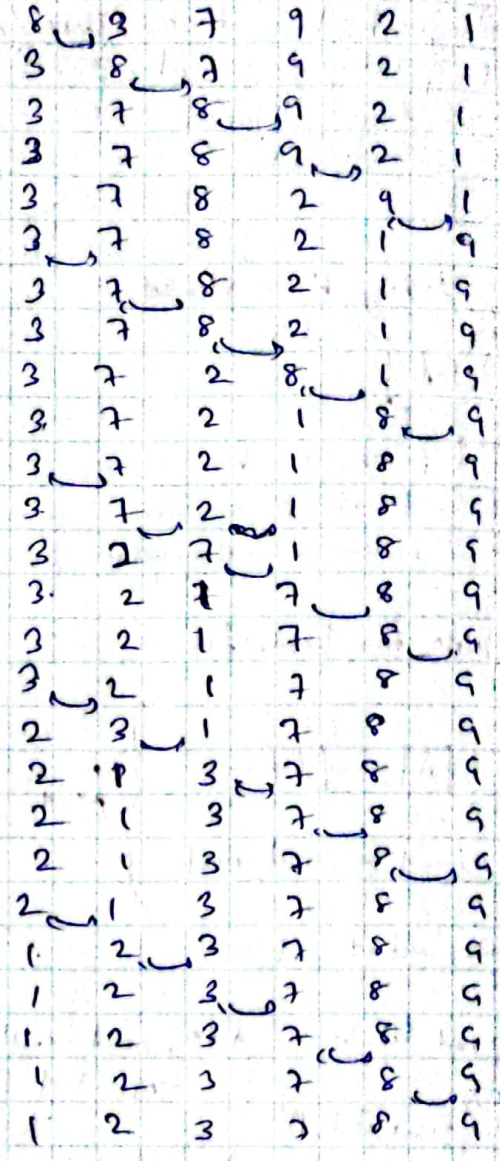
What is sorting?
↳ Sorting refers to the operation or technique of arranging and rearranging sets of data in some specific order.

Different types of sorting algorithms:
↳① Bubble Sort
↳② Insertion Sort
↳③ Selection Sort
↳④ Merge Sort
↳⑤ Quick Sort

① Bubble Sort
↳ which continually swaps nearby components if they are in the wrong order.

Large data sets should not be used with this approch due to its high average and worst case time complexity.

Q)




```
int arr[5] = {8, 3, 7, 9, 2, 1};
int j, temp;
int i = 0;
int a = arr[j]
for(int ; <6; ++){
    for(int ; <6; ++){
      if (a[i] > arr[i]) {
          temp = arr[j];
      arr[j] = arr[i];
          arr[j] = temp;
      }
    }
}
```

```
int a = 5;
int b = 6;
int temp;

int a = temp; temp = a;
    a = b;
    b = temp;
```

```
for i <- 0 to list: Length - 1          ← enhancing algorithm
for j <- 0 to list: Length - 1 [-i]
  if list[j] > list[j+1]
     Swap(list[j], list[j+1])
```

Eg 3. 4. 1. 5 6

```
for i <- 5 to list: Length - 1
  for j <- 5 to list: Length - 1 - i
```

3	4	1	5	6
3	4	1	5	6
3	1	4	5	6
3	1	4	5	6
1	3	4	5	6
1	3	4	5	6
1	3	4	5	6

⊛ Stable / Unstable Algorithms.

× Stable Algorithms.

→ If two items with equal keys appear in the output data set in the same order as they do in the input data set, a sorting algorithm is described as stable.

Sorting Algorithms

1) Bubble Sort
2) Insertion Sort
3) Selection Sort

① Insertion Sort.

8	2	10	5	6
0	1	2	3	4

→ (2)

8		10	5	6
0	1	2	3	4

Status :

8	2	10	5	6
0	1	2	3	4

Step 1 : ~~Take the first value of the first index and compare it with~~ (temporary variable) Take the value of the first index and compare it with value of the zeroth index value.

Step 2 : If it is greater than to value, then move the value of the zeroth index to first index.

Step 3 : Then insert the value of the temporary variable into the zeroth index.

Step 4 : Then take the value of the second index, compare it with sorted items, according to closest so. In this example with first index if it is greater than the value simply move and the insert the temporary value to previous index.

Step 5 : Likewise do this, until getting a sorted list

2	5	6	8	10
0	1	2	3	4

Algorithm

To sort an array of size N in ascending order:

→ Iterate from arr[1] to arr[N-1] over the array

Compare the current element (key) to its predecessor.
If the key element is smaller than its predecessor compare it to the elements before.
Move the greater elements one position up to make space for the swapped element.

Pseudo Code

```
mark the first element as sorted.
For each unsorted element X {.
'extract' the element X
for(j= lastSorted Index down to 0){
    if (current element j>X){.
        move sorted element to the.
        right by 1.
    }.
    breaks the loop and insert X here
}
}
```

→ Where to use Insertion Sort?

- Used when number of elements is small
- Can be useful when input array is almost sorted
- When only few elements are misplaced.
- In complete big array.

② Selection Sort

This sort an array by repeatedly finding the minimum item (when talking about ascending order) from the unsorted part and putting it in front

The algorithm have two sub-arrays in a given array.

- The sub-array which already sorted.
- The remaining sub-array was unsorted

→ Define two arrays.
→ One array that hold the unsorted array.
→ Define the 0th index value as sorted.
→ Then compare with ~~first one, if it is larger than~~ minimum value of the unsorted array if it is ~~higher than~~ smaller than the sorted one. push it to the front.
→ do this again until sorted the array

Algorithm

- Initialize minimum value (minindex) to location 0.
- Traverse the array to find the minimum element in the array.
- While traversing if any element smaller than minindex is found then swap both the index values.
- Swap elements
- Then, increment minidex to point to the next element.
- Repeat until the array is sorted.

Pseudo Code

```
N = Length of the array Arr
for i <-0 to (N-1):
    min_index = i
    for j <- i+1 to (N-1):
        if Arr[min_index] > Arr[j]:
            min_index = j
    Swap (Arr[i], Arr[min_index])
```

When to use Selection Sort?

- On short arrays, it works really well
- Is useful for determining whether everything has already been sorted.
- When memory utilization should be considered. This is due to the fact that selection sort uses less temporary storage space since, in contrast to other sorting algorithms it saves time by only switching items at the very end.

Recursion

Lecture Game.

Algorithm

Ask and wait
  Ask and wait
    Ask and wait
      Reach first in line Tell person behind
      it's 0
    Tell person it is a +1
  Tell person it is x +1

What is Recursion?

↳ Recursion is the process of defining a problem (or the solution to a problem) in terms of (a simpler version of) itself

Two main components of a recursive algorithm

↳ ① Base Case
  ↳ The point where you stop applying the recursive case

↳ ② The Recursive Case
  ↳ The set of instructions that will be used over and over.

Pseudo Code

Definition: A recursive function is a function that calls itself.

```
numberOfPeopleInfront(person){
  If (there is no one to tap)
    return 0

  Else
    ask the person in front of you (P)
    #ppl infront of P: numberOfpeopleFront(P)
    return #ppl in front of P+1
}
```

Fibonacci → 1, 1, 2, 3, 5, 8, 13, 21.

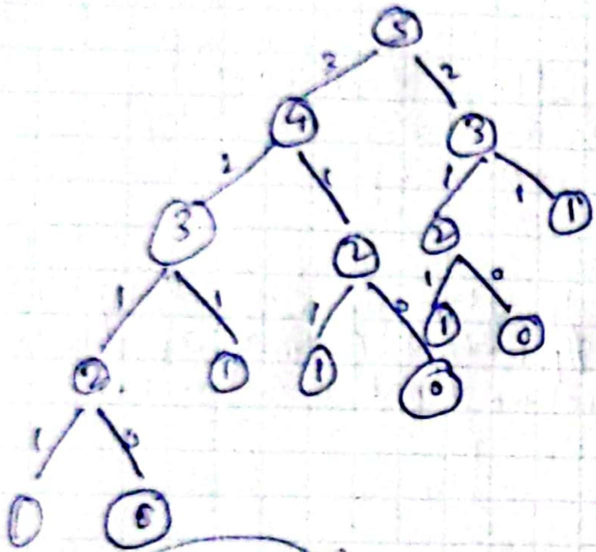
```
fib (n){
  take first two numbers as 1 and 1
  add two numbers = new number
  count equal to 2
  [add new number with the
   previous number,
   count = count +1.        ← recursive case
  loop runs until count equal to
  6.                        ← Base case
  return the value

  z=0;
  x=1;
  y=1;
  new_number = x+y;
  count = 2;
  for (int i= count =2; count<6;
  count ++)
```

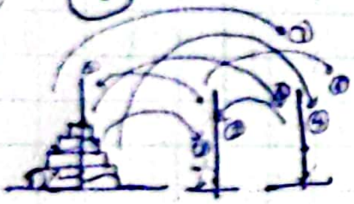
Jan-20 - 2110

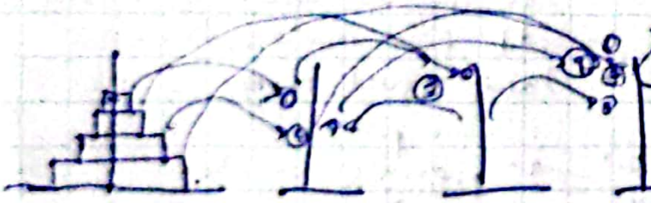
x = new x + y ;
y = new x - y .

Algorithm

```
function: fib(n):
    if (n==0 or n==1):
        return 1
    return fib(n-1) + fib(n-2)
```

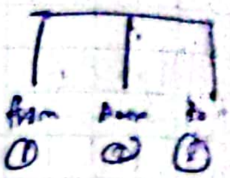


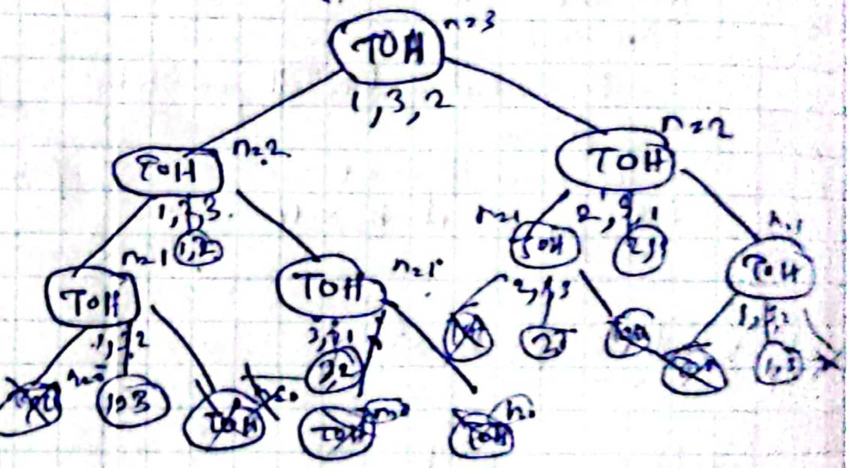




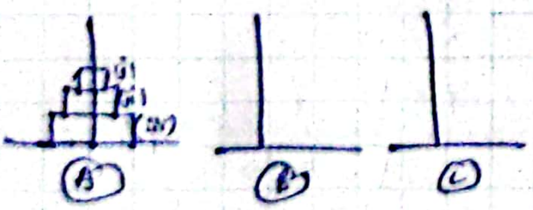
Step 1: ① to B.
② to C
① to C
③ to B
① to A
② to B
① to B

Step 2: ① to B.
② to C.
① to C
③ to B
① to A
① to B
① to B
④ to C
① to A
② to C
① to B.





Algorithm




```
function TowerOfHanoi (n, f_rod, t_rod, aux_rod)
    if n == 0;
        return.

    TowerOfHanoi (n-1, f_r, aux_rod, t_rod)
    print ("Move disk", n "from", f_rod, "to" t_rod)
    TowerOfHanoi (n-1, aux_rod, t_rod, f_rod)
```

Sorting oea - assignment - 1.5 hour/n hour - language - c

Sorting Methods.

(1) Internal Sorting Algorithms

↳ When all data is placed in the main memory or internal memory then sorting is called internal sorting.

Ex: heap sort, bubble sort, selection sort, quick sort, shell sort, insertion sort.

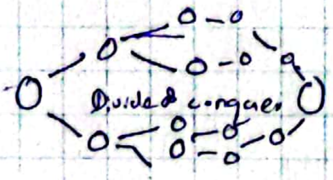
(2) External Sorting Algorithms.

↳ When the data being sorted do not fit into the main memory of a computing device (usually RAM) and instead they must reside in the slower external memory (usually the hard drive)

Ex: Merge Sort.

(3) Merge Sort

basic concept.



↳ algorithm depend on divide and conquer concept.

↳ divide the array into two equal parts until it becomes empty or one element left sort them seperately and finally join them together

Merge Sort - Pseudo Code

```
merge-sort (int A[], int start, int end)
{
    if (start < end) {
        int mid = (start + end)/2;  (defines the current array in 2 parts)
        merge-sort(A, start, mid);  (sort the 1st part of array)
        merge-sort (A, mid+1, end);  (sort the 2nd part of array)

    // merge both parts by comparing elements
    of both parts.
    merge (A, start, mid, end);
    }
}

merge(int A[], int start, int mid, int end) {

// stores the starting position of both parts in temporary array.

int p = start, q = mid+
```

A Quality Notebook Product