Date: / /

Complexity of Algorithm

Time Complexity

Space Complexity

Provided the data — n = 1000, n = 10000

speed doesn't matter — n = 100000

1- reduce number of steps

2- simplify the steps

D.S :

collection of data in some form

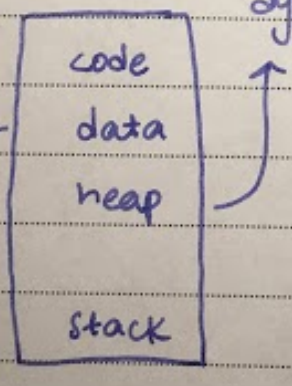
LIFO

FIFO

allocate, reallocate, release

↑

dynamic memory



- Array
- Stacks
- Linked Lists
- Queues
- Trees
- Graphs

insert
search
delete
sort
access

initialized, uninitialized data

data structures is not a hypothetical concept.

time and space complexity

Abstract Data Types

interface

public functions

Insert
Add car
Delete
Update
Search

implementation

ADT

data storage in a composite type data structure

collection of data

Programs use the interface to interact with the ADT.

owner of ADT knows inner/private details.

classes ⟶ objects

Date:

Data Structures

ADT Programs — Adam Drozdek and Algo 4th edition

- easier to write, read and modify.
- Focus on "what" the module does not "how". — Data Structures and Program design in C++
- Build loosely coupled functions
- Functional Abstraction
- Information hiding. What to hide?
- not only hide, make it inaccessible
- Hide implementation details from others

Data Structures

efficiency of operations

class is a method used to express both data and operations

Why ADT?

Composite data as one element of collection of data items

Virtuality — operations

Date: - -

```
resize_array(int *arr, int factor ){

{
    if (i == max_length) {

        *p = new int(2*max_length);

    }
```

stack

recursion

recursive search

```
search(*array, val, int nexindex) {
    if (arr[nextindex] == val) {
        return next_index;
    else if (nextindex == 0)
        return -1;
    else
        return search(i_arr, val, nextindex - 1);
```

Theory 3.67
Lab 4
Date:

Comparisons b/w two algos

number of operations varying with input.

ignoring the operations that have a constant time at which they are performed.

selection sort

```
n-1 times
    n-1 times
        min_index = j;
swap(a[i], a[min_index]);
```

Date: - -

Lecture # 7

```
int * arrresize (int *arr, int factor){
    int *temp = new (factor* size)

for(int i= 0; i< 2*size ; i++){
    temp[i] = arr[i];

    return tempi
}
```

will we delete the previous array?

how to avoid resize's cost

update
search
insert — shift right
delete — shift left

2D array
sort
give address of the searched element
row major format and column major format.

Lecture # 8 Date: - -

Linked Lists head → link → info = 101;

- declaration
- Head, tail, current pointers to manage lists.
- Traversing the list
- Insert ✓
- Update
- Delete

declaration
data type? of p?

```
struct node {
    int info;
    node *link;
};
```

```
node p = new node;
p → info = 101;
p → link = nullptr.
head → link = p;
```

Read number from keyboard?

```
cin >> num;
while (num != -1) {
    p = new node;
    p → info = num;
    p → next = nullptr;
```

Date: / /

```
cin >> num
while (num != -1)
    if (head == null) {
        head = new node;
        head -> info = num;
        head -> next = nullptr;
    }
    else {
        head -> next = new node;
        heal
```

(1) insert a new value after current ?

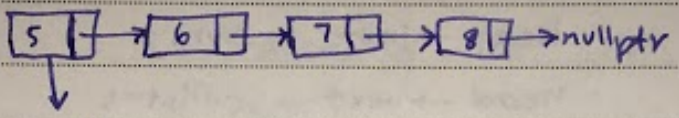
```
insert_after_current (node_t *c, int num) {
    node_t *t;
        t = new node_t;
        t -> info = num;
        t -> next = c -> link;

    return;
}
```

Sir can compromise on ~~coef~~ efficiency but not on correctness.

Date: - -

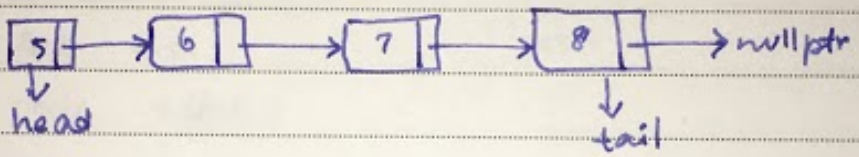
can linkedlists be inherited?

Singly Linked List



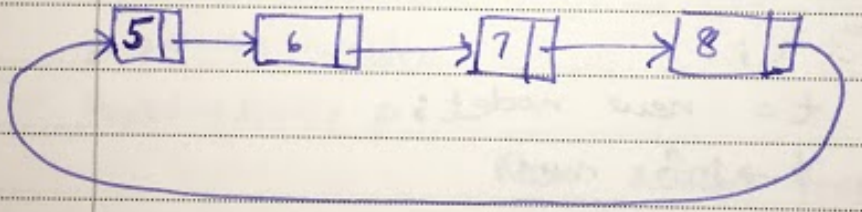
head

Doubly Linked List



tails in circular linked list?

Circular Linked List



insert

update

search

delete

sort

Date:

print a linked list

```
void print(){
    t = head;
    while (t != nullptr){
        cout << t->info << " ";
        t = t -> link;
    }
}
```

merge two linked lists.

recursive

```
~~void~~ pr  int print (node * current){
        if (t == nullptr)
            return 0;
        else {
            cout << t -> info << " ";
            ~~print(t~~
            print (current -> next);

print-recursion (node *p){      //reverse method
    if(p != nullptr){
        print-recursion(p -> next);
        cout << p -> info << " ";
    }
}
```