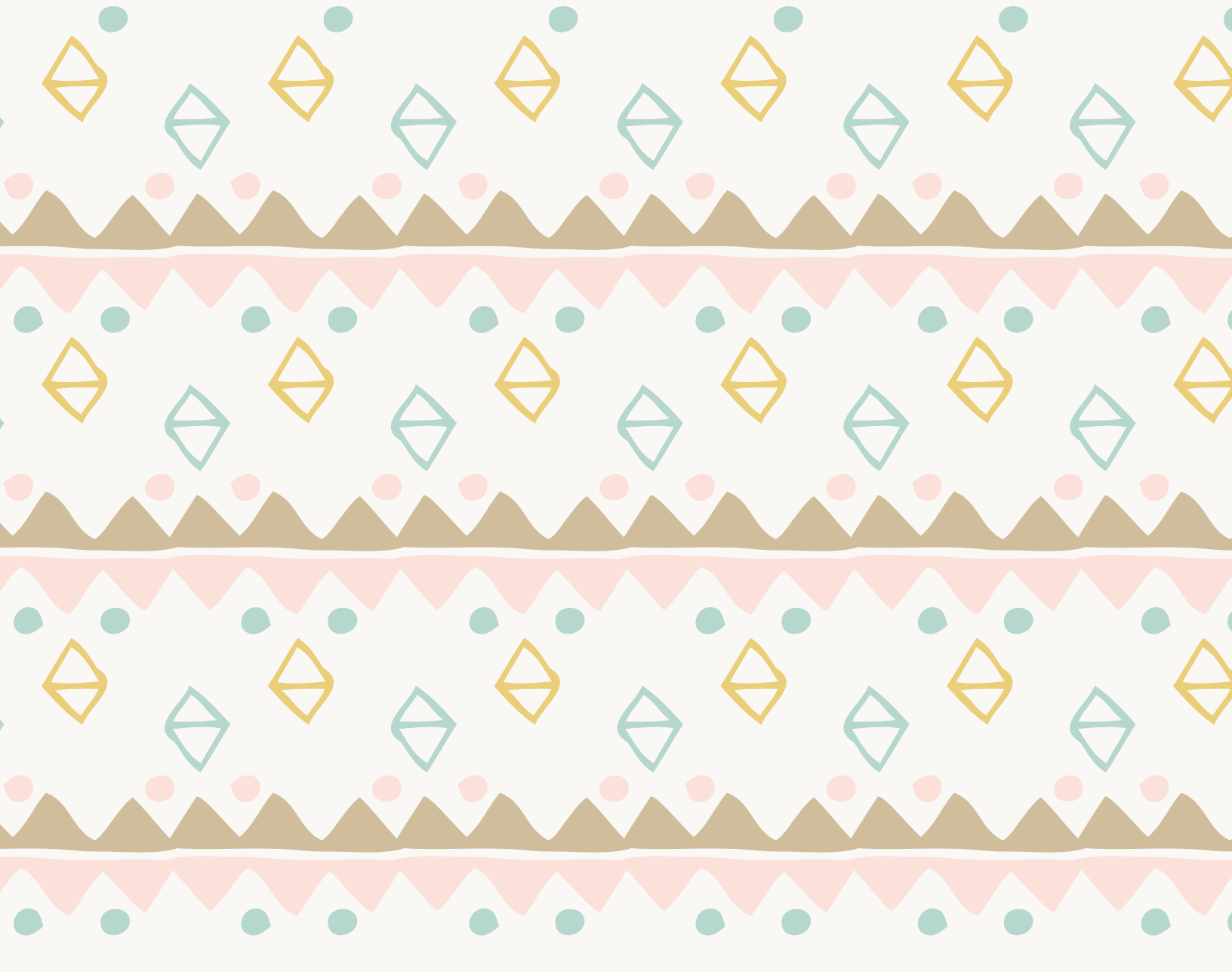


Recursive functions

```
int Fib (int n) {
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;

    return Fib(n-1) + Fib(n-2);
}
```

or

```
if (n <= 1)
    return n;
```

```
int main()
{
    Printf("Enter a number");
    int x;
    Scanf("%d", &x);
    Printf("The Fibonacci series of %d is %d", x, Fib(x));
    return 0;
}
```

Example

Fibonacci series

position:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Value:	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987

Code

C Fib.c > main()

```c
#include<stdio.h>
int Fib(int n)
{
    if (n<=1)
        return n;
    return Fib(n-1) + Fib(n-2);
}
int main()
{
    int x;
    printf("Please enter a number:\n");
    scanf("%d",&x);
    printf("The Fibonacci series of %d is %d \n", x, Fib(x));
    return 0;
}
```

Output

```
→  COMP-1410 gcc Fib.c
→  COMP-1410 ./a.out
Please enter a number:
3
The Fibonacci series of 3 is 2
→  COMP-1410 ./a.out
Please enter a number:
10
The Fibonacci series of 10 is 55
→  COMP-1410 ./a.out
Please enter a number:
16
The Fibonacci series of 16 is 987
→  COMP-1410 ./a.out
Please enter a number:
11
The Fibonacci series of 11 is 89
→  COMP-1410
```

Stack calls

C stackcalls.c > ...

```c
#include<stdio.h>
void blue()
{
    printf("inside blue\n");
    return;
}
void green()
{
    printf("inside green\n");
    return;
}
void red(){
    printf("inside red\n");
    green();
    printf("inside red from green\n");
    blue();
    printf("inside red from blue\n");
    return;
}
int main (){
    printf("inside main\n");
    red();
    printf("inside main from red\n");
    return 0;
}
```

Output

```
→  COMP-1410 gcc stackcalls.c
→  COMP-1410 ./a.out
inside main
inside red
inside green
inside red from green
inside blue
inside red from blue
inside main from red
→  COMP-1410
```

Pass by value

C pass_by_value.c > ...

```c
#include<stdio.h>
void inc(int x)
{
    ++x;
    printf("The value of x in inc is %d. \n", x);
}
int main()
{
    int x=1;
    printf("The value of x in main is %d. \n", x);
    inc(x);
    printf("The value of x in main is %d. \n", x);
}
```

Output

```
→  COMP-1410 gcc pass_by_value.c
→  COMP-1410 ./a.out
The value of x in main is 1.
The value of x in inc is 2.
The value of x in main is 1.
→  COMP-1410
```

Pointers

The address operator (&) produces the location of an identifier in memory (the starting address of where its value is stored).

- The printf format specifier to display an address (in hex) is "%p"

Syntax

- float *p; // pointer declaration
- p = &i ; // p now stores the address of i

```
printf("The values after swapping are: \nx = %.1f \ny = %.1f \n", x, y);// prints the values 5.5 and 8.1
printf("The Value addresses after swapping are:\nx = %p \ny = %p\n ", p1, p2); // prints 0x16d7634c8 and 0x16d7634c4
printf("The Pointer addresses after swapping are:\nx = %p \ny = %p\n ", &p1, &p2); // prints 0x16d7634c4 and 0x16d7634c4
```

```
int i = 42;
int * p = &i;

i => 42
&i => 0xF020
p => 0xF020
&p => 0xF024
```

- In C, we can define a pointer to a pointer:

  int *p1 = &i; // pointer p1 point at i

  int **p2 = &p1; // pointer p2 point at pointer p1.

- A void pointer (void *) can point at anything, including a void pointer (itself).

- Null is a special value that can be assigned to a pointer to represent that the pointer points at "nothing"

  int *p = NULL; // Good practice

- NULL is considered "false" when used in a Boolean context.

  Eg: The following are equivalent:

  → if (p) - - - .

  → if (p != NULL) - - - -

- The size of a pointer is always the same size that is 64 bits or 8 bytes in 64-bit environment

- The value of *&i or *&*&*&i is simply the value of i

- Code

```
int i =5;
int j= 6;
int *p= &i;
int *q = &j;
printf("P points at %p\n", p);
printf("Q points at %p\n", q);
p = q; // This is a pointer assignment. It means “change p to point at what q points at”.
printf("Afterwards\n P points at %p\n", p);
printf("Q points at %p\n", q);
```

Output

```
→  COMP-1410 ./a.out
P points at 0x16af274bc
Q points at 0x16af274b8
Afterwards
 P points at 0x16af274b8
Q points at 0x16af274b8
```

- The Statement

  *p = *q

```
int i = 5;
int j = 6;
int *p = &i;
int *q = &j;
printf("P points at %d\n", *p);
printf("Q points at %d\n", *q);
*p = *q; // it changes the value of i to 6 even though i was not used in the statement.
printf("Afterwards\n P points at %d\n", *p);
printf("Q points at %d\n", *q);
```

Output

```
→  COMP-1410 ./a.out
P points at 5
Q points at 6
Afterwards
 P points at 6
Q points at 6
```

- To pass the address of x use the address operator (&x)
- A function must never return an address to a variable within its stack frame.
  eg return &n; ✗ Never do this

Arrays

- The following two different are equivalent:
  - char a[3] = {'c', 'a', 't'};
  - char b[3] = "cat";
- Pointer notation
  - sum += *p; is the same as sum += a[i];
  - Sum += *(*(arr+k)+j); is the same as sum += arr[k][j];
- Const int *p: you can mutate p but you cannot mutate *p.
- Initialization of 2D arrays:

int two_d_arrays[3][6] = {{3,1,4,1,5,9},{2,6,5,3,5,8},{9,7,9,3,2,3}}

Reversing a number using recursion

Code

```
void reverseNumber(int num)
{
    // Base case: if the number is 0, just return
    if (num == 0)
    {
        return;
    }

    // Print the last digit
    printf("%d", num % 10);

    // Recursive call with the number divided by 10
    reverseNumber(num / 10);
}
```

* To generate random values and store them in an array

  -> first, #include <time.h>

  -> in the main, ① srand(time(0)); // Seed the random number generator.

  ② arr[i][j] = rand() % 100;

  generate numbers between 0 and 100

* To make an executable file in the terminal.

  gcc filename.c -o programname -lm.

* $\sum_{k=j}^{N} O(1) = (N-j+1)\,1$

Selection Sort

the smallest element is selected to be the first element in the new sorted sequence and then the smallest element is selected to be the second element and so on.

```
void selection_sort (int a[], int len)
{
    int pos = 0;
  for (int i=0; i< len-1; i++)
   {
      pos = i;
      for (int j=i+1; j< len; ++j)
       {
          if (a[j] < a[pos])
            {
              pos = j;
            }
          swap (&a[i], &a[pos]);
       }
   }
```

Insertion Sort

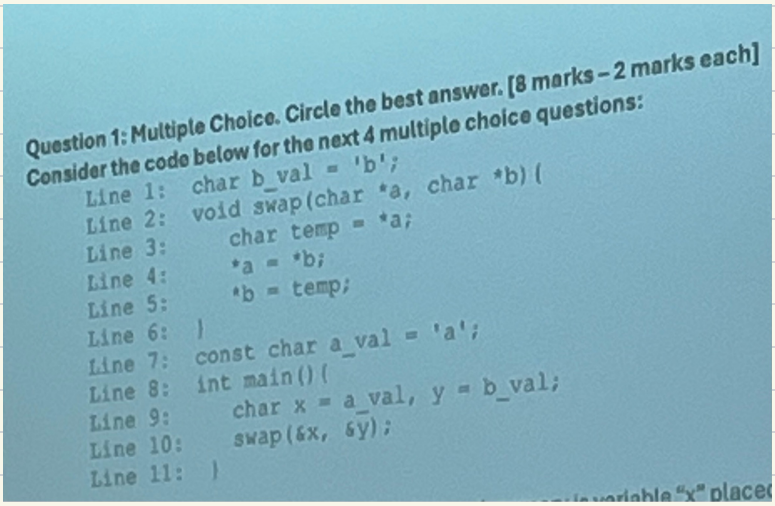
Big O notation

The "dominant" term is the term that grows the largest when n is very large ($n \to \infty$). The order is also known as the "growth rate"

i.e. $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$

if for example, there is $n^3 + 2^n = O(2^n)$ we consider (2^n) the dominant term

Midterm Correction

① 

Question 1: Multiple Choice. Circle the best answer. [8 marks – 2 marks each]
Consider the code below for the next 4 multiple choice questions:

```
Line 1:  char b_val = 'b';
Line 2:  void swap(char *a, char *b){
Line 3:     char temp = *a;
Line 4:     *a = *b;
Line 5:     *b = temp;
Line 6:  }
Line 7:  const char a_val = 'a';
Line 8:  int main(){
Line 9:     char x = a_val, y = b_val;
Line 10:    swap(&x, &y);
Line 11: }
```

variable "x" place

(A) x is placed in stack

(B) Read only

② (A) Char *P = &A;

(B)
```
int main()
{
    int array[20][20] = { };
}
```

(C)
```
int fact(int N){
    if (N==0)
        return 1;
    return N* fact(N-1);
}
```

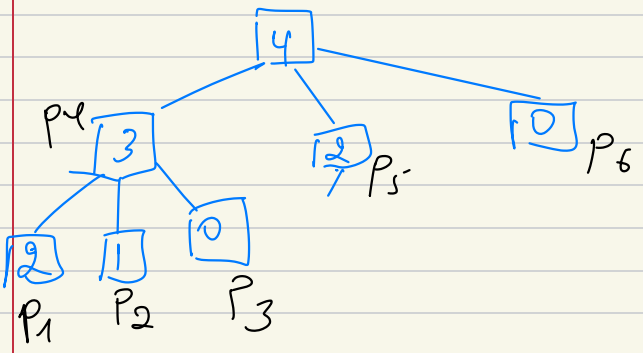
② (A)
```
for (int i=0 ; i<N ; i++)
    swap (&A[i] , &B[i]);
```

(E)
```
for( int i=0 ; i<N ; i++)
    swap (A+i , B+i);
```

③ (A) Binary search

```
int binary (int A[], int item, int L, int H)
{
    if (L > H)
        return -1 ;
    int M = (L+H)/L ;
    if (A[M] == item)
        return M ;
    if (A[M] > item)
        return binary(A, item, L, M-1);
    return binary(A, item, M-1, H);
}
```

(4) Stack



(5) (A)
```
void rotate(int N, int QR[ ][N]) {
    int copy[N][N];
  for (int i = k=0 ; k<N ; k++)
     for (int j=0 ; j<N ; j++)
        copy [k][j] = QR [j][N-1-k] ;
  for (int k=0 ; k<N ; k++)
    for (int j=0 ; j<N ; j++)
       QR[k][j] = copy [k][j];
}
```

* &QR[k][j]; in pointer Notation is QR + k * N + j;

* the inside pixels are (+2) eg N+2 while counting.

* for function parameters, the first parameter gets ignored by the compiler eg: N[3][4]; This means that it is fine to leave it blank.

* scanf and printf scan and print everything.

for example

example: scanf and buffers

```
int main(void) {
  int balance = 0;
  char command[8];
  while (1) {
     printf("Command? ('balance', 'deposit', or 'q' to quit): ");
     scanf("%s", command);
     if (strcmp(command, "balance") == 0) {
       printf("Your balance is: %d\n", balance);
     } else if (strcmp(command, "deposit") == 0) {
       printf("Enter your deposit amount: ");
       int dep;
       scanf("%d", &dep);
       balance += dep;
     } else if (strcmp(command, "q") == 0) {
       printf("Bye!\n"); break;
     } else {
       printf("Invalid command. Please try again.\n");
     }
  }
}
```

→ Here if you enter 99999999999. it will override and allocate the extra (9999) to the variable balance. which is a loophole.

Heap

The heap is the final section in the C memory model. This memory is "borrowed" and can be "returned"

Advantages of heap-allocated memory

* Dynamic
* Safety
* Resizable
* Duration

low

Code
Read-Only Data
Global Data } Constants or Global
Heap ⇒ borrow (must returned)
↓
↑
Stack ⇒ Inside function

High

malloc (memory allocation)

The malloc function obtains memory from the heap. it is proxided in <stdlib.h>

note Casting: float b = 1.5;

int a = (int) b;

Casting.

For example: int * my_array = malloc (100 * sizeof(int));

* There is also a calloc function which essentially calls malloc and then initializes the memory by filling it with zeros. Calloc is O(n) where n is the size of the block.
* We assume malloc is O(1).
* A common error is to forget to free allocated memory. (MUST_DO)

```
int * my_array = malloc (n * sizeof (int));
//---
//---
free (my_array);
```

* The heap allows us to return an array.

```
int * func(int n)
{

    int *array = malloc(n * sizeof(int));
    for (int i = 0; i<n ;i++)
    {
        array[i] = i;
    }
    return array;
}
```

```
int main()
{
    int *a = func(5);
    for (int i = 0; i < 5; i++)
    {
        printf("%d\n", a[i]);
    }

    free(a);
```

Recurrence relations

The recurrence relations we encounter in this course are:

$T(n) = O(1) + T(n-c)$	$= O(n)$
$T(n) = O(n) + T(n-c)$	$= O(n^2)$
$T(n) = O(n^2) + T(n-c)$	$= O(n^3)$
$T(n) = O(1) + T(\frac{n}{d})$	$= O(\log n)$
$T(n) = O(1) + d \cdot T(\frac{n}{d})$	$= O(n)$
$T(n) = O(n) + d \cdot T(\frac{n}{d})$	$= O(n \log n)$
$T(n) = O(1) + T(n-c) + T(n-c')$	$= O(2^n)$

where $c, c' \geq 1$ and $d > 1$

This table will be provided on exams if necessary.

Procedure for recursive functions

1. Identify the order of the function *excluding* any recursion
2. Determine the size of the data for the next recursive call(s)
3. Write the full *recurrence relation* (combine step 1 & 2)
4. Look up the closed-form solution in a table

```
int sum(int n) {
  if (n == 0) return 0;
  return n + sum(n - 1);
}
```

1. All non-recursive operations: $O(1)$ `+, -, ==`
2. size of the recursion: $n - 1$
3. $T(n) = O(1) + T(n-1)$ (combine 1 & 2)
4. $T(n) = O(n)$ (table lookup)

Insertion Sort

```cpp
#include <iostream>
using namespace std;

int main()
{
    int arr[5] = {5,2,6,3,5};
    int i, j , temp ;
    for(i = 0 ; i < 5 ; i++){
        temp = arr[i];
        j = i - 1;

        while(j>= 0 && arr[j] > temp){
            arr[j +1] = arr[j];
            j--;
        }
        arr[j+1] = temp;
    }
    for(int i = 0 ; i < 5 ; i++){
        cout << arr[i] ;
    }
    return 0;
}
```

Linked Lists

A linked list is a data structure storing a linear collection of data elements.

Example Problems and Solutions

Example 1: Create the structs required to define a linked list of integers.

```
struct Node {
    int data;
    struct Node* next;
};

struct LinkedList {
    struct Node* head;
};
```

Example 2: Use typedef.

```
typedef struct Node {
    int data;
    struct Node* next;
} Node;

typedef struct LinkedList {
    Node* head;
} LinkedList;
```

Example 3: Create a list with 2 nodes.

```
int main() {
    //This is without typedef
    struct LinkedList list;
    struct Node* first = malloc(sizeof(struct Node));
    struct Node* second = malloc(sizeof(struct Node));
    //This is with typedef
    //LinkedList list;
    //Node* first = malloc(sizeof(Node));
    //Node* second = malloc(sizeof(Node));

    first->data = 1;
    first->next = second;

    second->data = 2;
    second->next = NULL;

    list.head = first;

    return 0;
}
```

Linked list is on the final

→ Dynamic memory

preferably later content than beginning content

Note assert (q -> front) // throws an error if the front is Null

Tree terminology

- Root node has no parent
- Nodes can have multiple children.
- binary tree means that each node has 2 children.
- Leaf node has no children.
- Height of a tree (maximum possible number of nodes from the root node to a leaf node)
- Node count (number of node)

For Next time

Page 27 → lecture 9

Finals Prep

(1)

(X) printing a single character of the string, we use "%c" while when we want to include the whole string like "Hello world" we use "%s" but also do not include the brackets

(A)

```c
//Question 1
#include<stdio.h>
#include <string.h>
int main()
{
    char a[] = "Cat";
    char b[4] = "Cat";
    char c[] = {'C','a','t','\0'};
    char d[4] = {'C', 'a', 't', '\0'};
    printf("a = %s", a);
    printf("\nb = %s", b);
    printf("\n c = %s ", c);
    printf("\nd = %s", d);
    printf("\n");


    return 0;
}
```

```
→  Finals Prep
→  Finals Prep ./a.out
a = Cat
b = Cat
 c = Cat
d = Cat
```

(B)

```c
#include <stdio.h>
#include<string.h>
int main()
{
    char a[] = "Hello World";
    char b[13];
    strcpy(b, a);
    printf("B = %s\n", b);
    return 0;

}
```

```
→  Finals Prep gcc Finals.c
→  Finals Prep ./a.out
B = Hello World
```

1 C

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

int main()
{
    char str[12] = "Con";
    char array[] = "catenate";
    printf( "The first and the second string lead to = %s\n ", strcat(str, array));
    return 0;
}
```

```
➜  Finals Prep gcc Finals.c
➜  Finals Prep ./a.out
The first and the second string lead to = Concatenate
```

1 E

```c
//Question 1E
#include<stdio.h>
int my_strlen(char *str){

    int count= 0, i = 0;
    while(*(str + i) != '\0')
    {
        count++;
        i++;
    }
    return count;
}

int main()
{
    char a[] = "Python";
    printf("The length of the word Python is = %d\n", my_strlen(a));
    return 0;
}
```

```
➜  Finals Prep gcc Finals.c
➜  Finals Prep ./a.out
The length of the word Python is = 6
```

Reading a file

```
FILE *fp = fopen("filename", "r");
```

Writing a stack to a file.

```
bool stack_write(const char *filename, const struct stack *s);

bool stack_read(const char *filename, struct stack *s);
```

Writing a file

```
FILE *fp = fopen(filename, "w");
```

Close a file

```
int ret = fclose(fp);
```

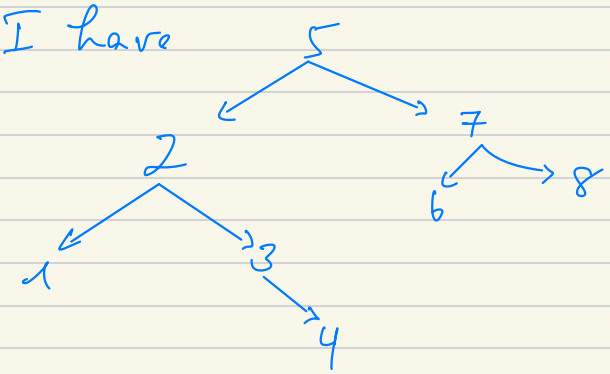
* fclose returns 0 on success then stores in "ret"

Appending a file

```
FILE *fp = (filename, "a");
```

TREE

if I have



1, 2, 3, 4 / 5, 6, 7, 8

Bubble sort

Compares two values next to each other.

Selection sort

finds the smallest element first, then puts it to the first position and so forth.....

Quick sort algorithm

divides the array into two groups, the one