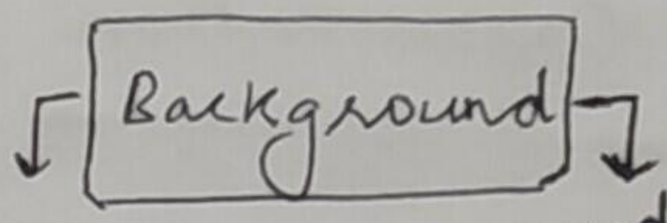
Runtime Environment



→ We write programs (C, C++, Java, Python), and save it in hard disk.

→ During compilation also program resides in hard disk

→ But CPU (processor) executes programs residing in main memory (RAM).

→ After compilation is over, the compiler demands a block of main memory from the OS, in order to store that program in main memory. (As OS manages the resources of computers, like CPU time, memory, Files, I/O devices, Disks, etc.)

→ OS allocates a free block of memory to the corresponding program, so the compiler uses that free block of memory in order to store that compiled program. This is called as Run-time Storage Management.

Storage Organization

(1) Static Allocation

(2) Dynamic Allocation
- Stack Allocation
- Heap Allocation

(1) Static Allocation :→ is allocation of memory during compilation. For example normal variables, arrays etc.

→ Once memory is allocated at compilation it is not possible to change its size during run-time (execution time)

Drawbacks

(1) We must know in advance regarding the size of an array (No. of elements to be stored in array)

(2) wastage of memory (in case of allocated more than size).

Ex. int a[100], only upto a[5] is used

(3) If less size is allocated, than later on can not be ~~increased~~ increased.

for ex. int a[5], and ~~one~~ no. of elements in array are 10, then last 5 elements of array can not be stored.

(4) Insertion or deletion operations are very expensive. (lot of shift operations needs to to performed)

---

(2) Dynamic Allocation { stack / Heap

→ Whenever a function call occurs, then an activation record will be created for the corresponding function.

→ Then that activation record will be pushed onto the stack (top of the stack)

→ If there are n functions in our program then n activation records will be created for corresponding n functions.

What is an activation record ?

→ Whenever a function (or procedure) call occurs then an activation record gets created, an activation record information is pushed onto the stack.

[Model of Activation Record
or
Field of Activation Record] → Mainly contains 7 fields

Actual Parameters
Returned values
Control or Dynamic Link
Access or Static Link
Saved Machine Status
Local Variables
Temporary Variables

① Actual parameters

→ The parameters which are declared inside the calling function.

```
c = sum(a, b);        int sum(int x,
                              int y)
```

Actual Parameters ⊆ = Formal Params

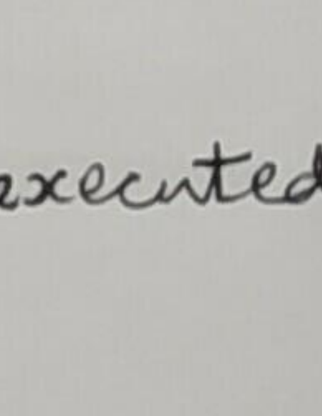
② Returned values

→ To store the result of function call.

③ Control or dynamic link

→ It points to the Activation Record of the calling function.

```
add(){
        sub();
}
```



④ Access or Static link

→ It refers to the local data of the called function but found in another Activation Record.

⑤ Saved Machine Status

→ Stores address of next instruction to be executed.

⑥ Local Variables

→ These variables are local to a function.

⑦ Temporary Variables

→ Needed during expression evaluation.

Drawback of Stack Allocation

- It supports dynamic allocation but slower than static allocation.

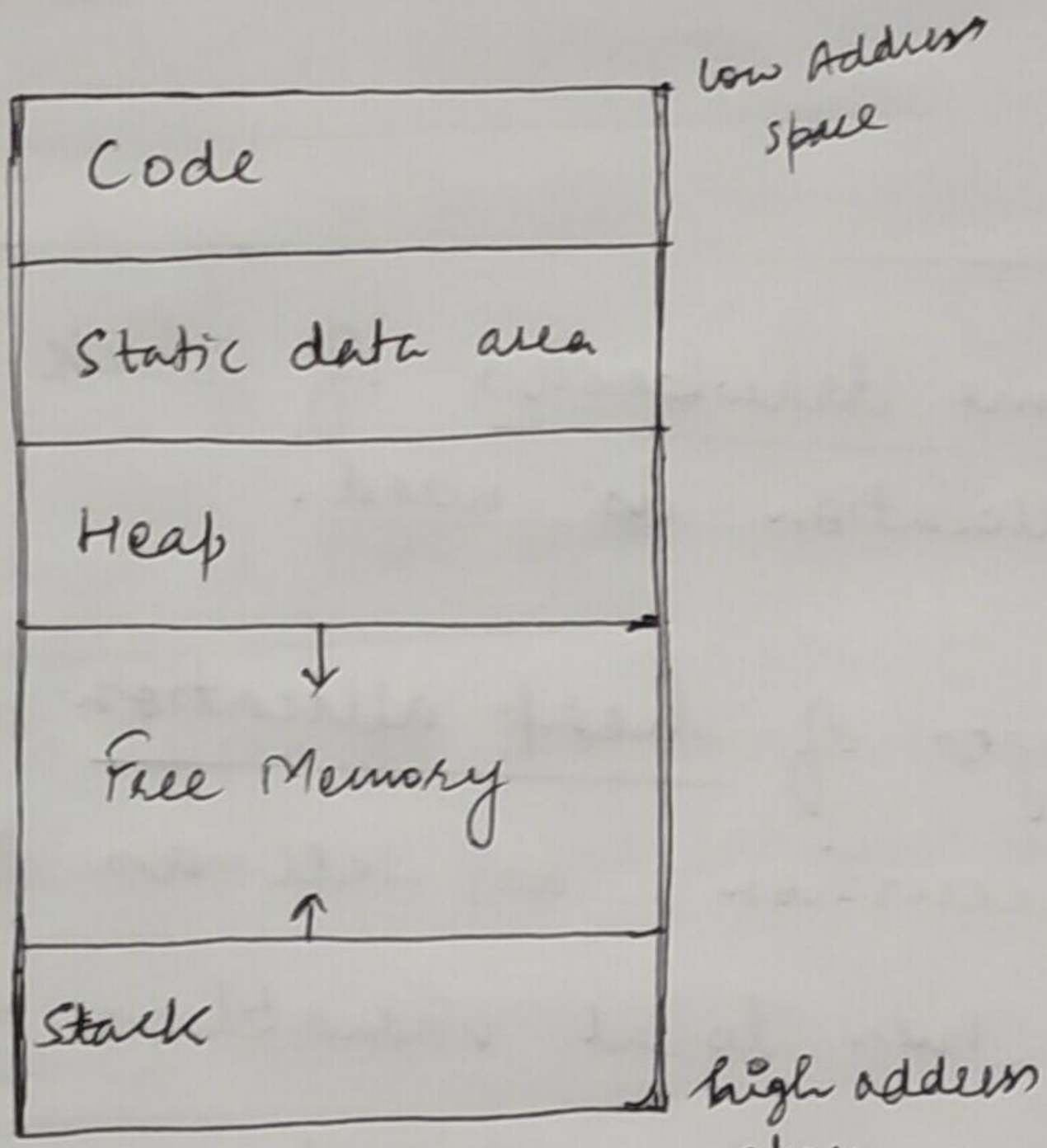
- It supports recursion but references to non-local variables after activation records can't be retained.

→ In order to overcome drawbacks of stack allocation, heap allocation is used.

→ The major advantages of heap allocation is it supports the recursion, as well as the references to the non local variables after activation records can be retained.

→ So mainly, we use heap allocation for dynamic memory allocation. In C programming there are functions like malloc(), calloc(), realloc(), free() } to allocate, resize and free memory blocks.

Typical subdivision of run-time memory into code and data areas



① Code

→ during compilation only the size of the program will be decided.

→ The code area stores the executable code here.

→ Linking links the object code of several library files into a single file, as executable file.

→ So basically code area contains target executable code.

② Static Data Area

→ It is mainly useful for storing static variables and global variables.

③ Heap and Stack

→ In order to effectively utilise space heap and stack are used.

→ stack grows from high portion to low portion (addres space)

→ Heap grows from low addres space to high addres space

Activation Trees

- An activation tree shows the way control enters and leaves activations.

- Properties of activation trees are :-

(1) Each node represents an activation of a procedure.

(2) The root shows the activation of the main function.

(3) The node for procedure 'x' is the parent of node for procedure 'y' if and only if the control flows from procedure 'x' to procedure 'y'

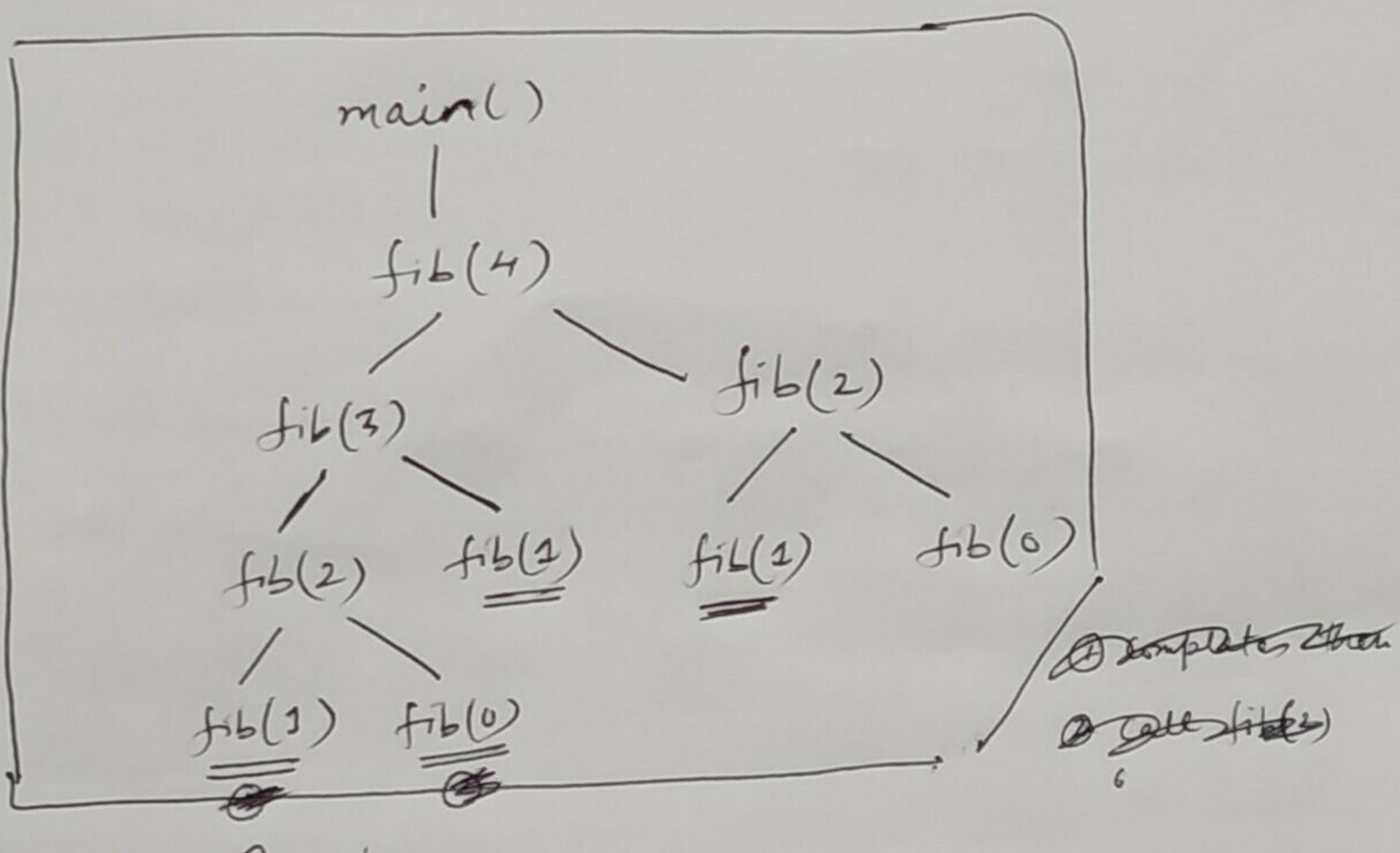
// Example – Consider the following program of (16)
Fibonacci :

```
main(){
  // int n;
  fib(4);
}
```

⇒

```
int fib(int n){
  if (n==0 || n==1)
    return 1;
  else
    return fib(n-1) + fib(n-2);
}
```

Activation tree will be (for fib(4))



✓ Recursive function
✓ Various activation of fib() is present

Try to write activation tree for Quicksort()

* [Please see geeksforgeeks.org]