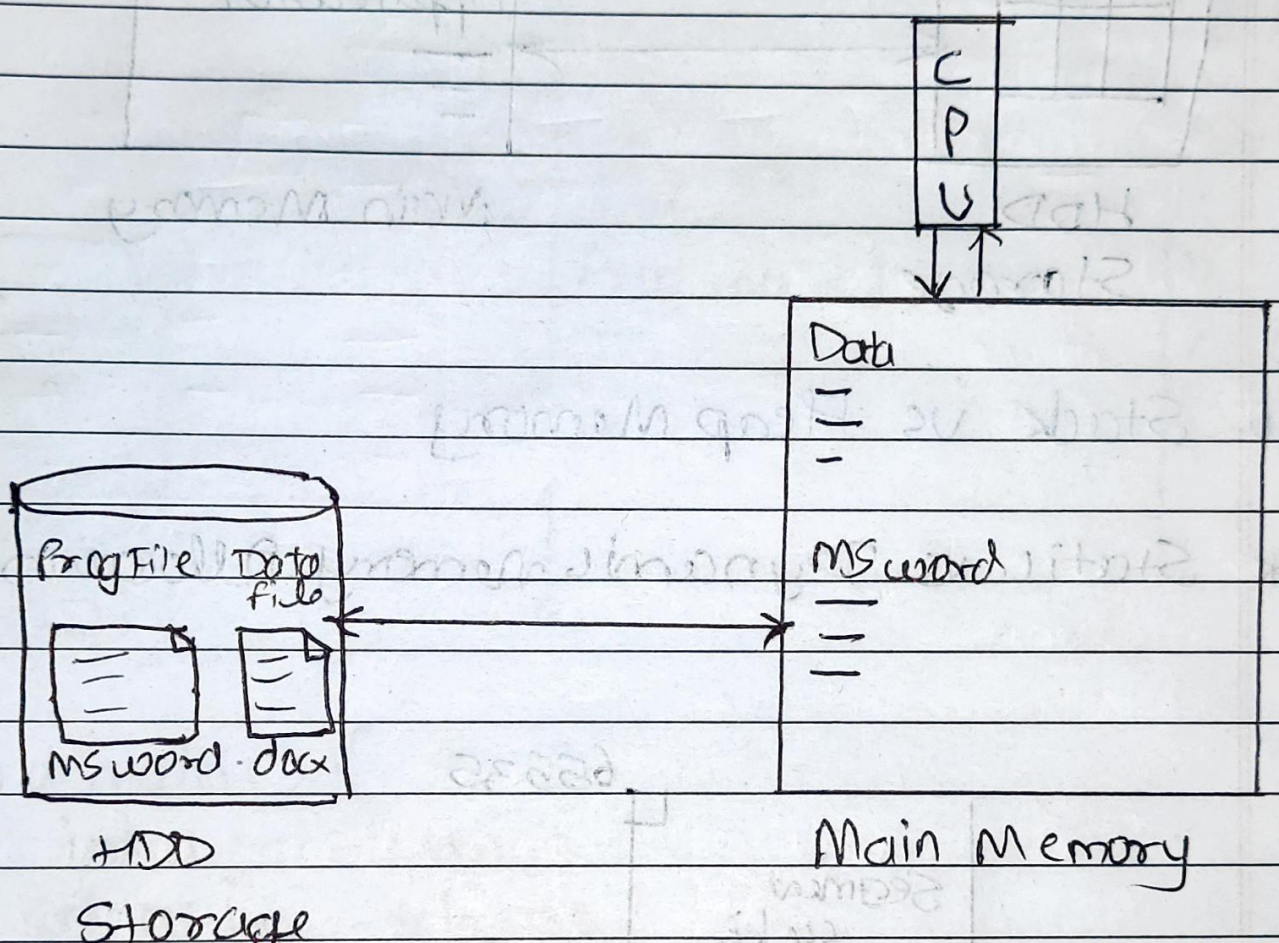
18/12/21

Section 4:

* Introduction

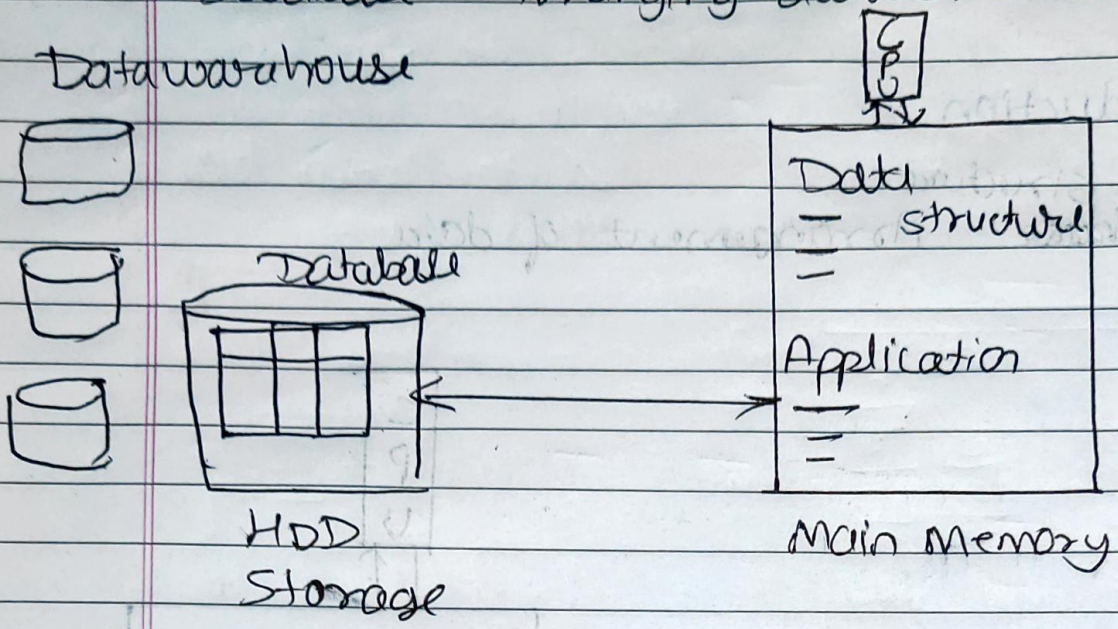
39. Introduction

* Database - Arrangement of data structure



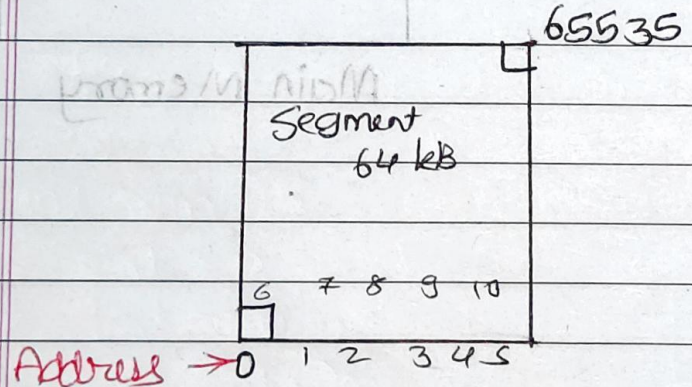
* Database - Arranging data in table

Datawarehouse



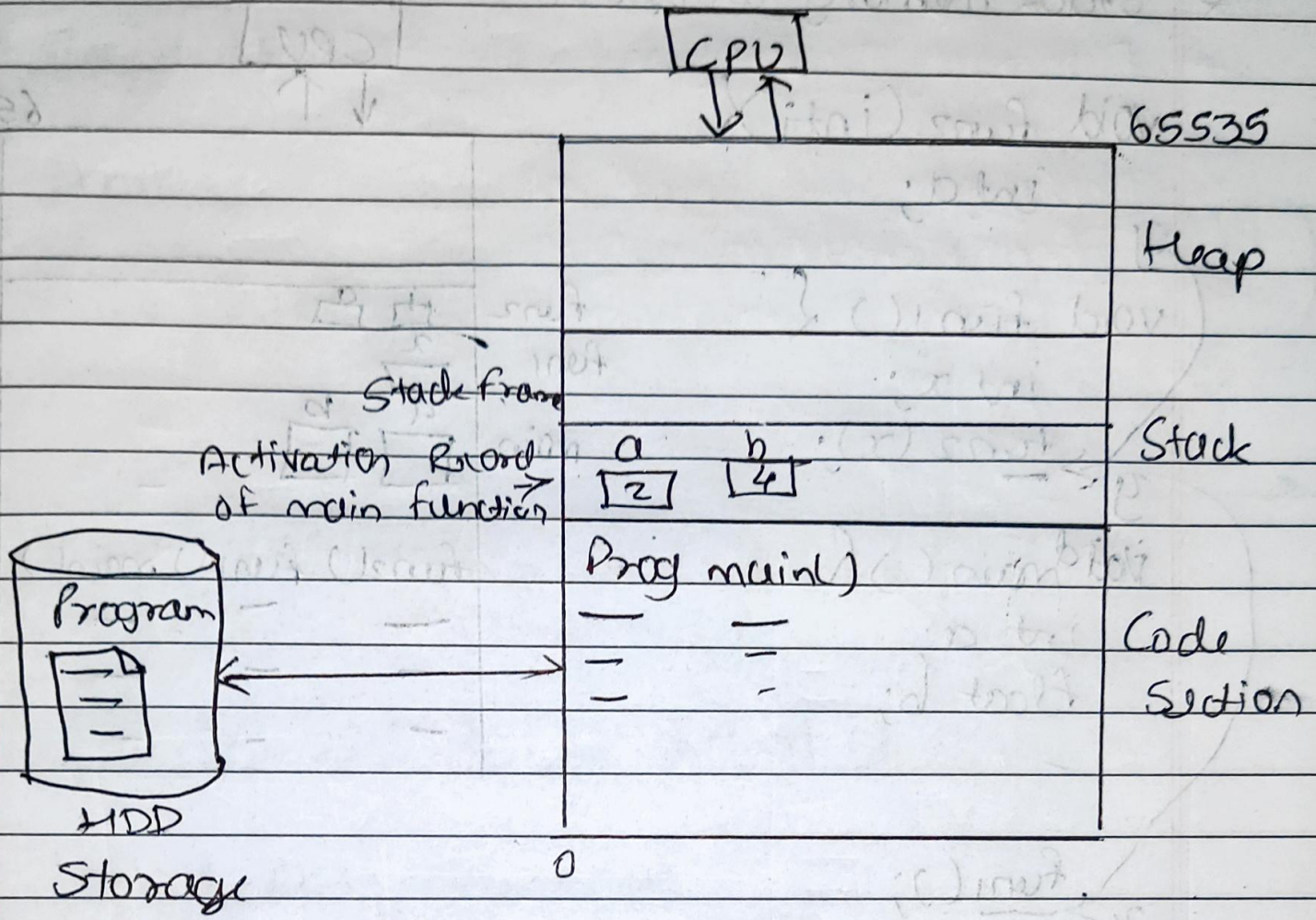
40. Stack vs Heap Memory

* Static vs Dynamic Memory Allocation



0 - 65535 = 655356 = 64 * 1024 = 64 kB

* How a Program uses main memory

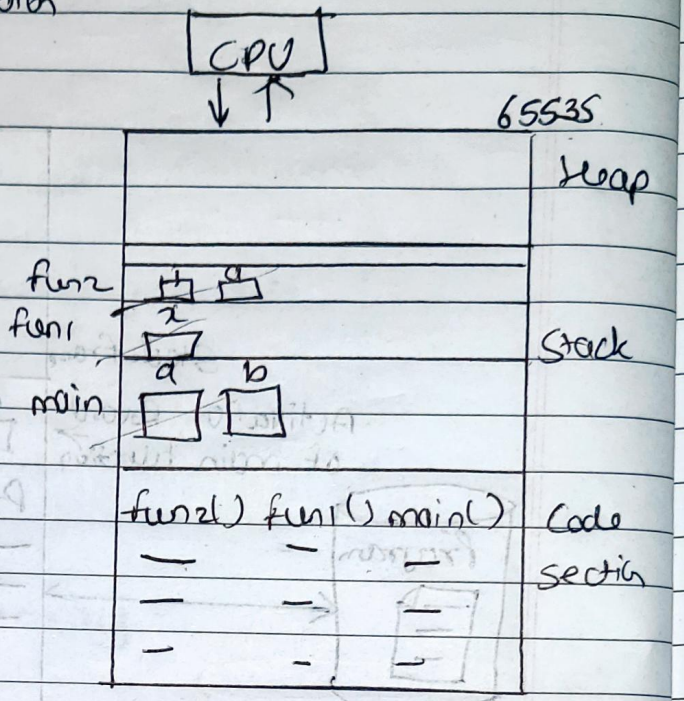



```
void main() {
    int a;      - 2 bytes
    float b;    - 4 bytes
}
```

41. Stack vs Heap continued

* Stack memory utilization

```
void fun2(int i){
    int a;
}
void fun1(){
    int x;
    fun2(x);
}
void main(){
    int a;
    float b;
    -
    -
    fun1();
}
```



* Heap memory utilization

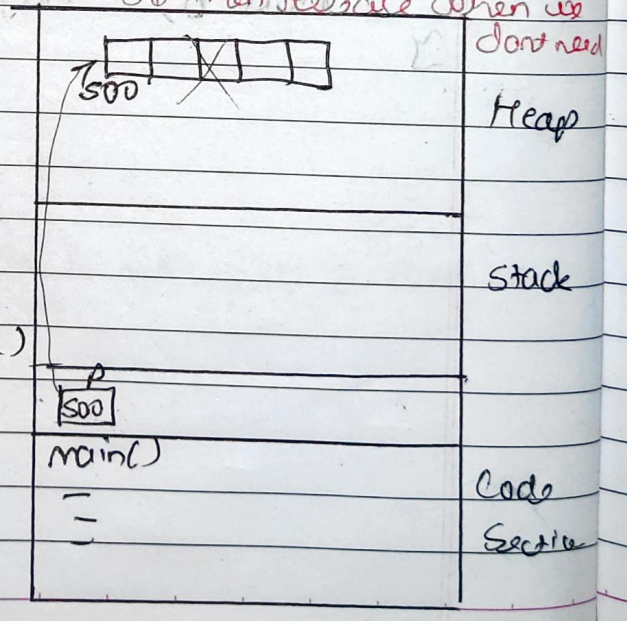
Heap → we using pointers and then release when we don't need

```
int main(){
    int *p; → 2 bytes
    p = new int[5];
    p = (int*)malloc(2*5);
    delete[] p;
    → p = NULL;
}
```

C++ →

C →

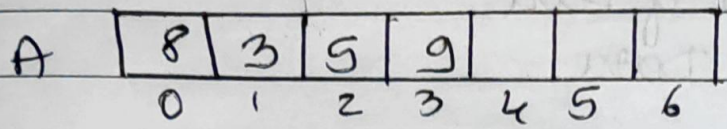
deallocate whenever don't need

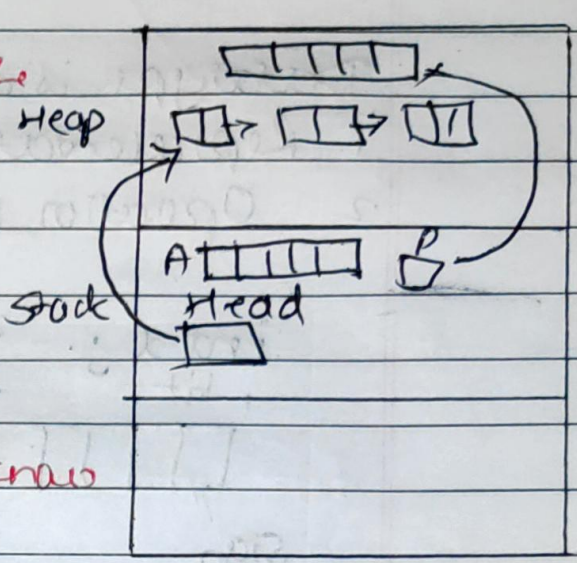


42 Physical vs Logical Data Structures

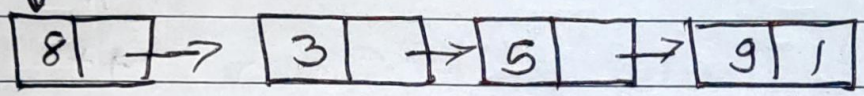
* Physical Data structures

1. Array → when we know the list





2. Linked List → when we don't know the list



* Logical Data Structures

1. Stack } Linear – LIFO
2. Queues } – FIFO
3. Trees } Non-Linear
4. Graph }
5. Hash table → Tabular

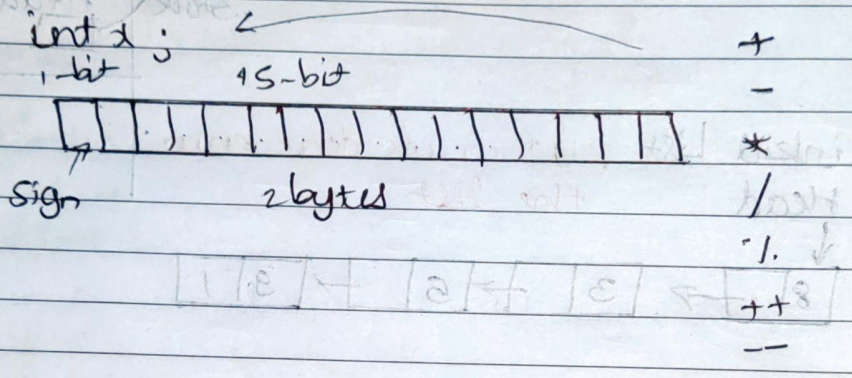
* Logical structures are implemented using physical Data structures

43. ADT

Hiding all internal details

* Abstract Datatype

Datatype is defined as
1. Representation of Data
2. Operation on Data



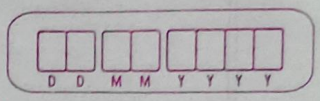
List => 8, 3, 9, 4, 6, 10, 12
0 1 2 3 4 5 6

Data : 1. space for storing element
2. Capacity
3. Size of List

use
1. Array
2. Linked List

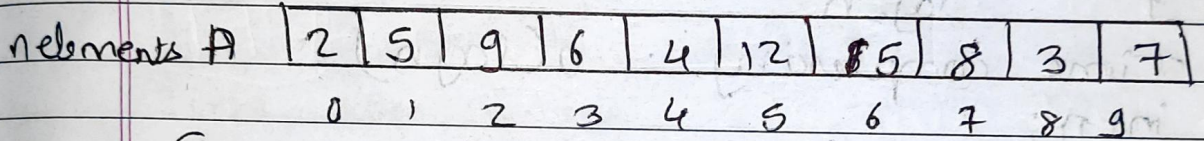
Operations on List :

add(x)
remove( )
search(key)
:



- add (element) / append (ele)
- add (index, ele) / Insert (index, ele)
- remove (index)
- set (index, ele) / replace (index, ele)
- get (index)
- search (key) / contains (key)
- Sort()

4. Time and Space Complexity



Space complexity : O(n)

```
* for (int i=0; i<n, i++) {
    ...
  }
```

n elements

Time complexity : O(n)

```
* for (int i=0; i<n, i++) {
    for (int j=0; j<n; j++) {
      ...
    }
  }
```

n × n elements

Time complexity : $O(n^2)$

* for(int i=0; i<n; i++){
  for(int j=i+1; j<n; j++){
  }
}

$1+2+3+\dots+n-2+n-1$

$$\frac{n(n-1)}{2} = \frac{n^2-n}{2} \approx n^2$$

Time complexity: $O(n^2)$

* Processing half list (divide by 2)

```
for(int i=n; i>1; i=i/2){
  ...
}
```

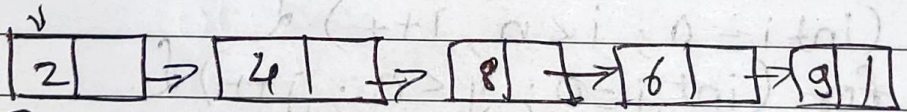
$\log_2 n$

Time complexity: $O(\log n)$

```
int n;
while (i>1){
  ...
  i=i/2;
}
```

Time complexity: $O(\log n)$

* first



Space complexity: $O(n)$

Same as Array

* A (4×4, n×n):

	0	1	2	3
0	8	3	5	9
1	7	6	4	2
2	6	5	3	9
3	10	4	2	6

n^2 elements

Time complexity : $O(n^2)$ → all elements

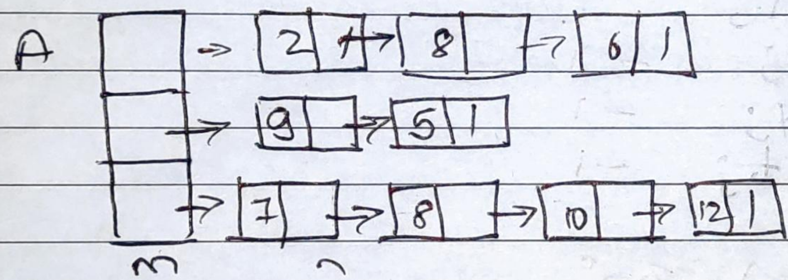
for row or column $O(n)$

Space complexity : $O(n^2)$

```
for(i=0; i<n; i++){
    for(j=0; j<n; j++){
        for(.....){
        }
    }
}
```

n^3

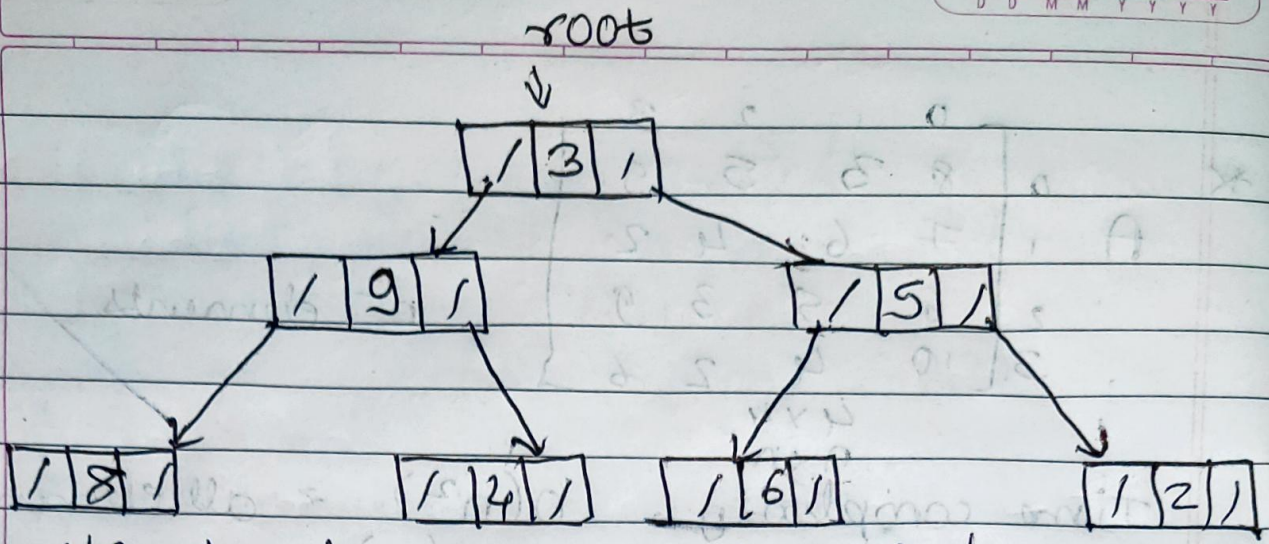
Time complexity : $O(n^3)$

* 

Time complexity : $O(m+n)$ → when considering both m and n

: $O(n)$ → when considering only n

Space complexity : $O(m+n)$



No. of elements are divided by two from bottom so logn

Time complexity : O(logn)

O(n)

Space complexity : O(n)

45. Time and Space complexity from code

1) void swap(x, y) {

int t;

t=x; — 1

x=y; — 1

y=t; — 1

}

$f(n) = 3n^0$

$O(1)$

$O(n^0) = O(1)$

Time complexity : O(1)

2) int sum (int A[], int n){
    int s, i;
    s=0; — 1
    for (i=0; i<n; i++){ — n+1 2(n+1)
        s= s+A[i]; — n
    }
    return s; — 1
}

$f(n) = 2n+3$

$O(n)$

Time complexity: $O(n)$

3) void add (int n){
    int i, j;
    for (i=0; i<n; i++){ — n+1
        for (j=0; j<n; j++){ — n × (n+1)
            c[i][j] = A[i][j] + B[i][j]; — n×n
        }
    }
}

$f(n) = 2n^2+2n+1$

$O(n^2)$

Time complexity: $O(n^2)$ → order of

Big $O(n^2)$

$\theta(n^2)$

$\Omega(n^2)$

4)
```
fun1() {
    fun2();
}

fun2() {
    for(i=0; i<n; i++) {
    }
}
```

O(n)

n

n

Time complexity: O(n)