

# Problem Solving Through Programming in C

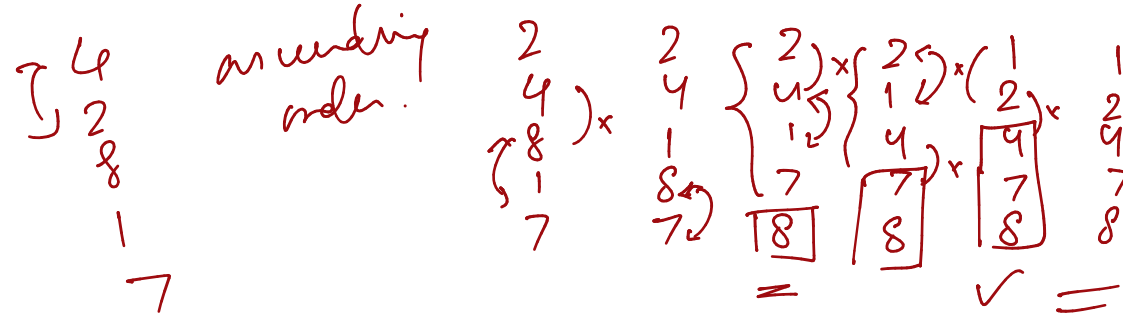
## Tutorial Session 9

---

Prof. Anupam Basu  
Dept. of Computer Science & Engg.  
IIT Kharagpur

Siddhant Mohapatra  
PMRF Scholar  
IIT Madras

# Bubble sorting



Q. How can you improve the best-case efficiency in bubble sort? (The input is already sorted)

a) `boolean swapped = false;`  
`for(int j=arr.length-1; j>=0 && swapped; j--)`  
`{`  
 `swapped = true;`  
 `for(int k=0; k<j; k++)`  
 `{`  
 `if(arr[k] > arr[k+1])`  
 `{`  
 `int temp = arr[k];`  
 `arr[k] = arr[k+1];`  
 `arr[k+1] = temp;`  
 `swapped = false;`  
 `}`  
 `}`  
`}`

b) `boolean swapped = true;`  
`for(int j=arr.length-1; j>=0 && swapped; j--)`  
`{`  
 `swapped = false;`  
 `for(int k=0; k<j; k++)`  
 `{`  
 `if(arr[k] > arr[k+1])`  
 `{`  
 `int temp = arr[k];`  
 `arr[k] = arr[k+1];`  
 `arr[k+1] = temp;`  
 `}`  
 `}`  
`}`

c) `boolean swapped = true;`  
`for(int j=arr.length-1; j>=0 && swapped; j--)`  
`{`  
 `swapped = false;`  
 `for(int k=0; k<j; k++)`  
 `{`  
 `if(arr[k] > arr[k+1])`  
 `{`  
 `int temp = arr[k];`  
 `arr[k] = arr[k+1];`  
 `arr[k+1] = temp;`  
 `swapped = true;`  
 `}`  
 `}`  
`}`

d) `boolean swapped = true;`  
`for(int j=arr.length-1; j>=0 && swapped; j--)`  
`{`  
 `for(int k=0; k<j; k++)`  
 `{`  
 `if(arr[k] > arr[k+1])`  
 `{`  
 `int temp = arr[k];`  
 `arr[k] = arr[k+1];`  
 `arr[k+1] = temp;`  
 `swapped = true;`  
 `}`  
 `}`  
`}`

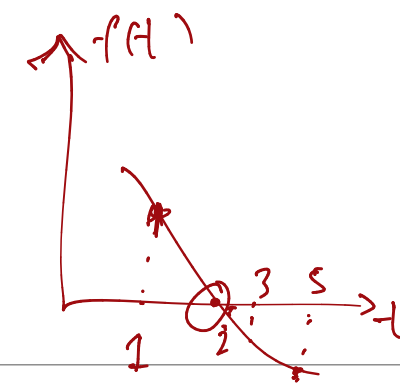
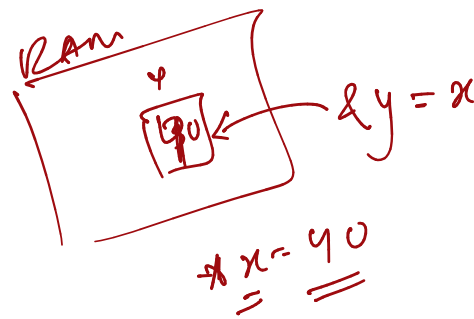
$j = 3$   
 $j \geq 0$  (false)  
 swapped = false  
 loop exits

No comp  
 $\sim O(N^2)$  4 comp.

best case  
 $O(N)$  time complexity  
 1st pass no swapping  $\Rightarrow$  swapped = false

$j = 5 - 1 = 4$  (true)  
 $j \geq 0$  & swapped  
 $k = 0$  to 3  
 $arr[0] > arr[1]$  false  
 $arr[1] > arr[2]$  "  
 $arr[2] > arr[3]$  "

# Miscellaneous



Q. What is the output of the following program?

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

```
void func(int x)
{
    x = 40;
}
```

```
int main()
{
```

```
    int y = 30;
    func(y);
    printf("%d", y);
    return 0;
}
```

- a) 40
- ☒ b) 30
- c) Runtime error
- d) Compilation error

Handwritten notes for Q1:  
 -  $x = 30$  (initial value)  
 -  $x = 40$  (change in func)  
 -  $(int\ x)$  (pass by value)  
 -  $func(30)$  (call from main)  
 -  $func(&y)$  (call by reference, but code shows  $func(y)$ )  
 - 40 (final value of x in func, not y in main)

Q. Assuming an initial range  $[1, 5]$ , the second (at the end of 2 iterations) iterative value of the root of  $te^{-t} - 0.3 = 0$  using the bisection method is (Note: you need to find the root, not the function value)

$$f(t) = te^{-t} - 0.3$$

$$f(1) = 0.067$$

$$f(5) = -0.266$$

$f(t) \rightarrow 0$  at what  $t$ ?

roots of the  $f^n$ .

$$f\left(\frac{1+5}{2} = 3\right) \text{ 1st iter.}$$

$$= f(3) = -0.151$$

$$f\left(\frac{1+3}{2} = 2\right) \text{ 2nd iter.}$$

$$= f(2) = -0.03$$

$$f(1.5) = \underline{\hspace{2cm}}$$

(2)

# Miscellaneous

Q. Find the output of the following program

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

```
int main()
```

```
{
    int *ptr, a = 12;
    ptr = &a;
    *ptr = *ptr - 2**ptr;
    printf("%d, %d", *ptr, a);
    return 0;
}
```

*ptr points to the location of a*  
*ptr → a memory location*

*12, -12*  
*output*

*\*ptr = 12*

*\*ptr = 12 - 2\*12 = -12*

Q. What is the solution of the equation given below using the Bisection Method up to four decimal places? (Consider the root lying on positive quadrant only and compute the root till five iterations only)

$$f(x) = xe^{2x} - 3x^2 - 5$$

*root after 5 iterations = 1.03125 ≈ 1.0312*

$$f(0) = -5$$

$$f(1) = -0.611$$

$$f(2) = 92.1962$$

*1st iter*

$$f(1.5) = 18.38$$

*2nd iter*

$$f(1.25) = 5.54$$

*3rd iter*  
 $f(1.125) = 1.877$

*4th iter*  
 $f(1.0625) = 0.509$

*5th iter*  
 $f(1.03125) = -0.079$

1.0312

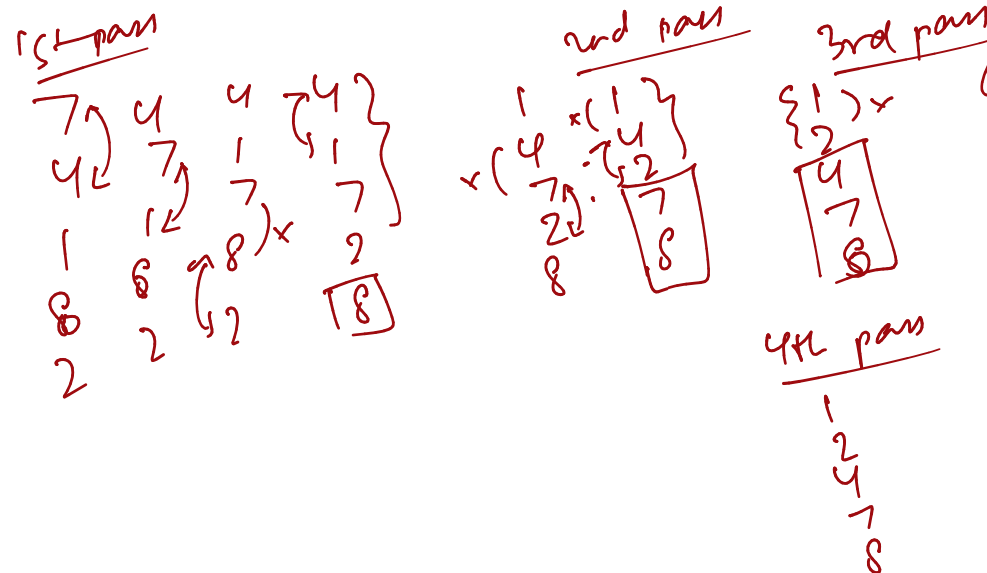
# Miscellaneous

pointer which points nowhere

dangling pointer.

Q. What are the correct intermediate steps of the following data set when it is being sorted with the bubble sort? 7,4,1,8,2 *ascending order*

- a) 4,7,1,8,2 → 4,1,7,2,8 → 4,1,2,7,8 → 1,4,2,7,8 → 1,2,4,7,8
- ✓ b) 4,7,1,8,2 → 4,1,7,8,2 → 4,1,7,2,8 → 1,4,7,2,8 → 1,4,2,7,8 → 1,2,4,7,8
- c) 4,7,1,8,2 → 1,4,7,8,2 → 1,4,2,7,8 → 1,2,4,7,8
- d) 4,7,1,8,2 → 4,7,1,2,8 → 1,4,7,2,8 → 1,4,2,7,8 → 1,2,4,7,8



Q. Which of the following statement is correct for the 2 arrays with respect to A and B.

```
int *x[5];
int *(y[5]);
```

- A. Array of pointers
- B. Pointer to an array

each element is a pointer  
pointer which points to the first element of an array

- a) x is A, y is B
- ✓ b) x is A, y is A
- c) x is B, y is A
- d) y is B, y is B

# Miscellaneous

Q. What will be the output?

```
#include <stdio.h>
int main(void)
{
    int a[] = {10, 12, 6, 7, 2};
    int i, *p;
    p = a + 4;
    for(i=0; i<5; i++)
        printf("%d ", p[-i]);
    return 0;
}
```

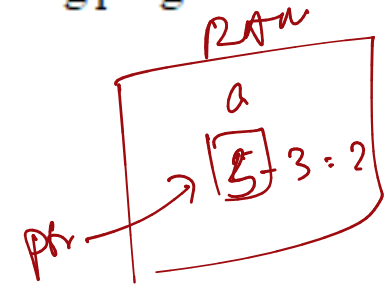
- a) 10 12 6 7 2
- b) 10 12 6 7
- c) 2 7 6 12
- ☒ d) 2 7 6 12 10

Handwritten notes for the first question:

array a: 10, 12, 6, 7, 2  
 indices: 0, 1, 2, 3, 4  
 p = a + 4 → points to index 4 (value 2)  
 p[-0] = \*p = 2  
 p[-1] = \*(p-1) = 7  
 p[-2] = \*(p-2) = 6  
 p[-3] = \*(p-3) = 12  
 p[-4] = \*(p-4) = 10

Q. Find the output of the following program

```
#include <stdio.h>
int main()
{
    int *ptr, a = 5;
    ptr = &a;
    *ptr = *ptr - 3;
    printf("%d,%d ", *ptr, a);
    return 0;
}
```



2, 2

# Miscellaneous

Sort in ascending order.

8  
7  
4  
2  
1

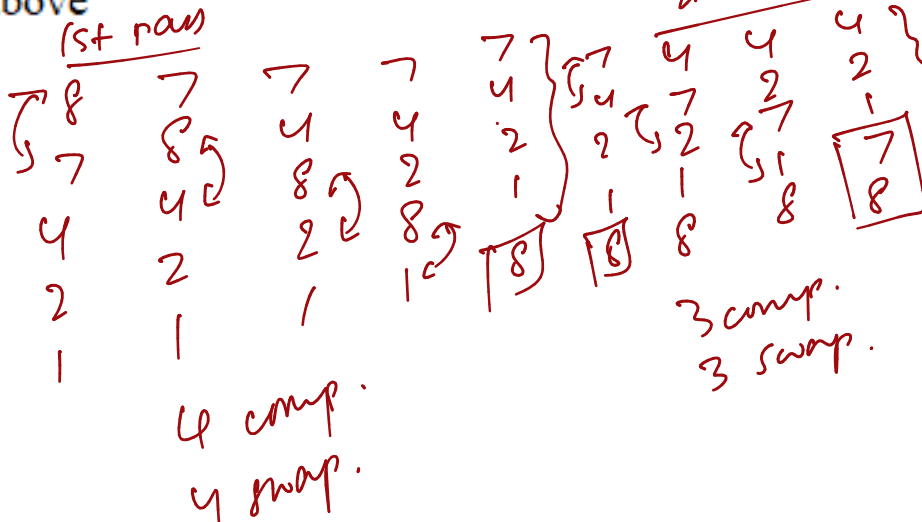
Worst case complexity  $\Rightarrow$  max no. of comparisons & swapping ops

Q. What maximum number of comparisons can occur when a bubble sort is implemented? Assume there are  $n$  elements in the array.

- a)  $(1/2)(n-1)$
- ✓ b)  $(1/2)n(n-1)$  +  $\frac{n(n-1)}{2}$  swaps
- c)  $(1/4)n(n-1)$
- d) None of the above

No. of ops  
 $\therefore 4+3+2+1$  comp  
 $+ (4+3+2+1)$  swap  
 $\therefore \frac{N(N-1)}{2} \times 2$

$\therefore N(N-1)$   
 $\sim O(N^2)$

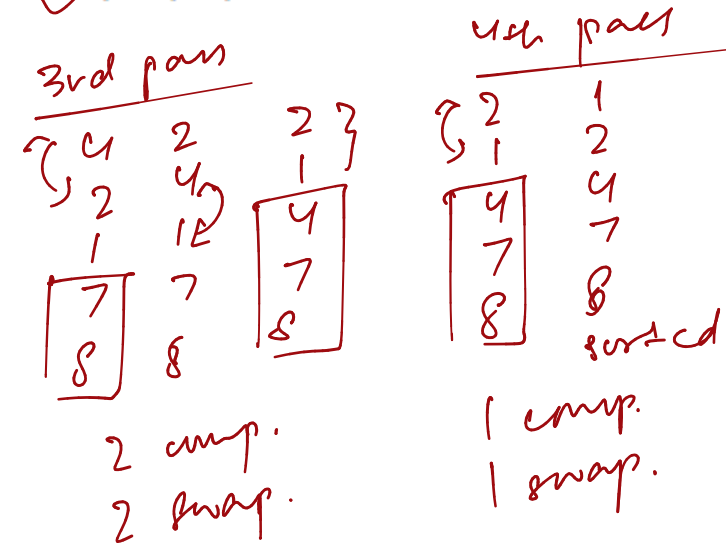


What is the worst-case complexity of bubble sort?

- a)  $O(N \log N)$
- b)  $O(\log N)$
- c)  $O(N)$
- ✓ d)  $O(N^2)$

best case  
 $\sim O(N)$

avg case  
 $\sim O(N^2)$





# Miscellaneous

*int \*x[7];  
int \*(y[7]);  
size of (y) = 56  
7x8*

What is the output of the code?

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

```
int main(){
```

```
    float (*x)[7][2];
```

```
    printf("%lu", sizeof(*x));
```

```
    return 0;
```

```
}
```

a) 14

b) 112

c) 28

☒ d) 56

*7x2x4  
= 56*

*→ pointer x to  
a float array  
of 7x2*

*sizeof(pointer) = 8 bytes  
64-bit m/c.  
32-bit m/c.  
16-bit m/c.*

*sizeof(float) = 4 bytes*

*sizeof(x) = 8 bytes.*

Q. What is the pointer expression for accessing arr[1][5][3]?

a) ((\* (arr+1)+5)+3)

b) (\*( (\* (arr+1)+5)+3)

☒ c) \* (\* (\* (arr+1)+5)+3)

d) \* (\* (\* (\*arr+1)+5)+3)

*arr[0][0][0] ⇒ \*\*\*arr*

*\* (\* (\* (arr+0+0+0) ) )*

*arr[1][5][3]*

*= \* (\* (\* (arr+1)+5)+3)*