Data Structure & Algorithms

Sunbeam Infotech

Agenda

- Q & A
- Recursion
- Binary Search (recursive)
- Quick Sort
- Merge Sort

1. if input array is already sorted, what is time complexity of insertion sort? → O(n)
2. if input array is already sorted, what is time complexity of selection sort? → O(n²)
3. if input array is already sorted, what is time complexity of further improved bubble sort?
4. num of comparisions for each sorting algo.
5. change your sorting codes to return num of swappings / shifting.
6. array of objects (students) & sort students by marks in desc. order.

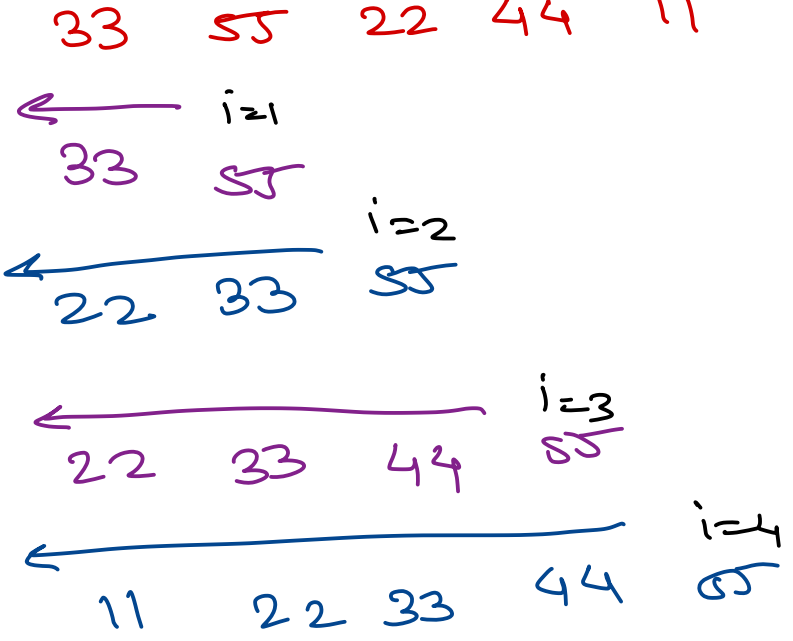
```
for(i=1; i<n; i++) {
  temp=a[i];
  for(j=i-1; j>=0 && a[j]>temp ; j--)
      a[j+1]=a[j];
  a[j+1] = temp;
}
```

11 22 33 44 55

11 22 33 44 55

itr = n-1

T ∝ n → O(n).



'insertion sort' best case time complexity = O(n).

```
for(i=1; i<n; i++) {
    flag = 0;
    for(j=0; j<n-1; j++) {
        if(a[j] > a[j+1]) {
            swap(a[j], a[j+1]);
            flag = 1;
        }
    }
    if (flag == 0)
        break;
}
```

11 22 33 44 55

compt++;

itr = n - 1

T ∝ n

O(n).

swapt++;
cnt

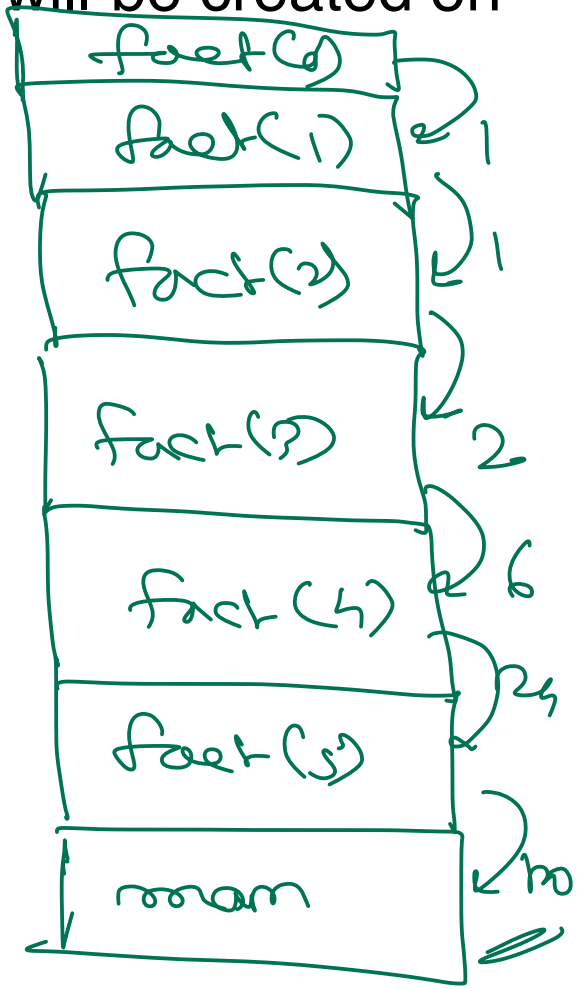
Recursion

- Function calling itself is called as recursive function.
- To write recursive function consider
  - Explain process/formula in terms of itself
  - Decide the end/terminating condition
- Examples:
  - n! = n * (n-1)! 0! = 1
  - $x^y = X * x^{y-1}$ $x^0 = 1$
  - $T_n = T_{n-1} + T_{n-2}$ $T_1 = T_2 = 1$
  - factors(n) = 1st prime factor of n * factors(n)

- On each function call, function activation record or stack frame will be created on stack.

```
int fact(int n) {
  int r;
  if(n==0)
    return 1;
  r = n * fact(n-1);
  return r;
}

res=fact(5);
```



$5! = 5 * 4!$

$4! = 4 * 3!$

$3! = 3 * 2!$

$2! = 2 * 1!$

$1! = 1 * 0!$

$0! = 1$

$2^4 = 2 * 2^3$

$2^3 = 2 * 2^2$

$2^2 = 2 * 2^1$

$2^1 = 2 * 2^0$

$2^0 = 1$

60 = 2 × factors(30)

30 = 2 × factors(15)

15 = 3 × factors(5)

5 = 5 × factor(1)

1

Binary Search

terminating condn.

30

$r < l$

0	1	2	3	4	5	6	7	8
11	22	33	44	55	66	77	88	99

bin search in array - l → r

① find mid of array.

② if key == arr[mid]
return mid;

③ if key < arr[mid]
bin search left part [l → mid-1]

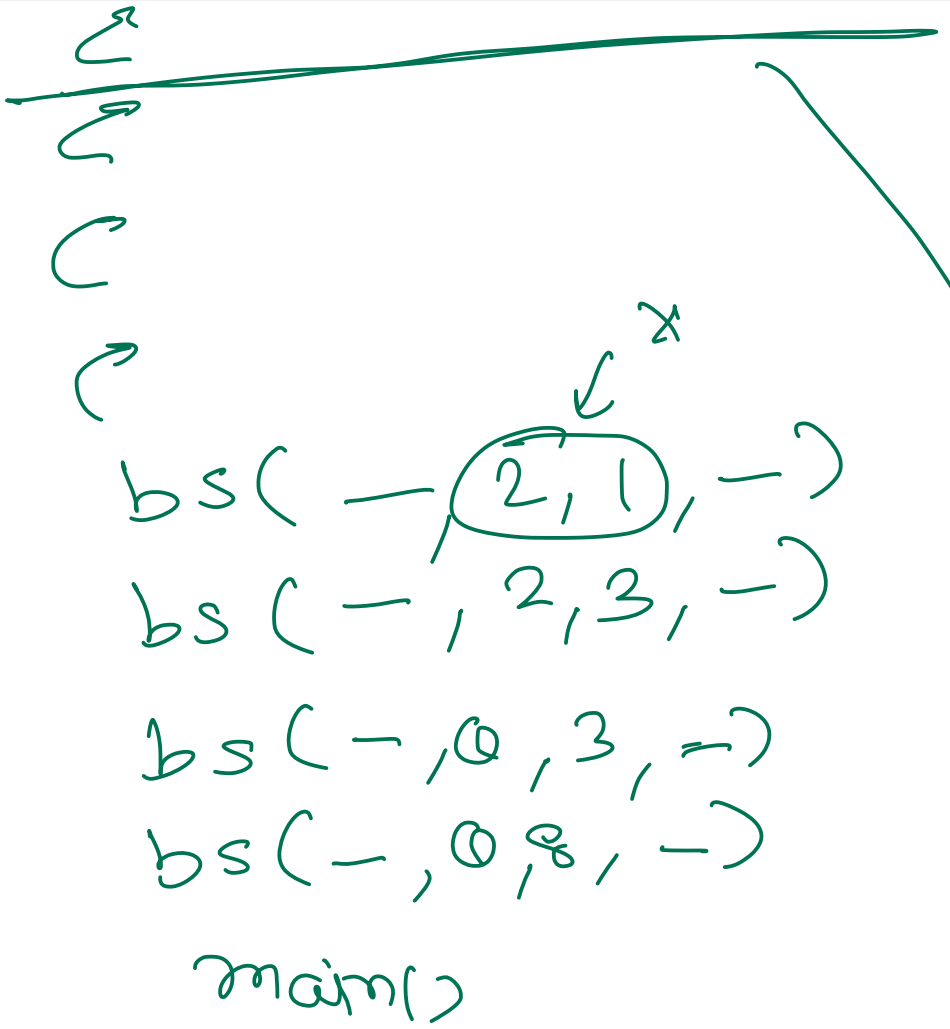
④ else // key > arr[mid]
bin search right part [mid+1 → r]

32
16 -
8 - → overheads
4 -
2 -
1 - → time complexity is same.

r space

```
int bin_search(int a[], int l, int r, int key){
    int m, i;
    if(r < l)
        return -1;
    m = (l+r)/2;
    if(key == a[m])
        return m;
    if(key < a[m])
        i = bin_search(a, l, m-1, key);
    else
        i = bin_search(a, m+1, r, key);
    return i;
}
```

Binary Search

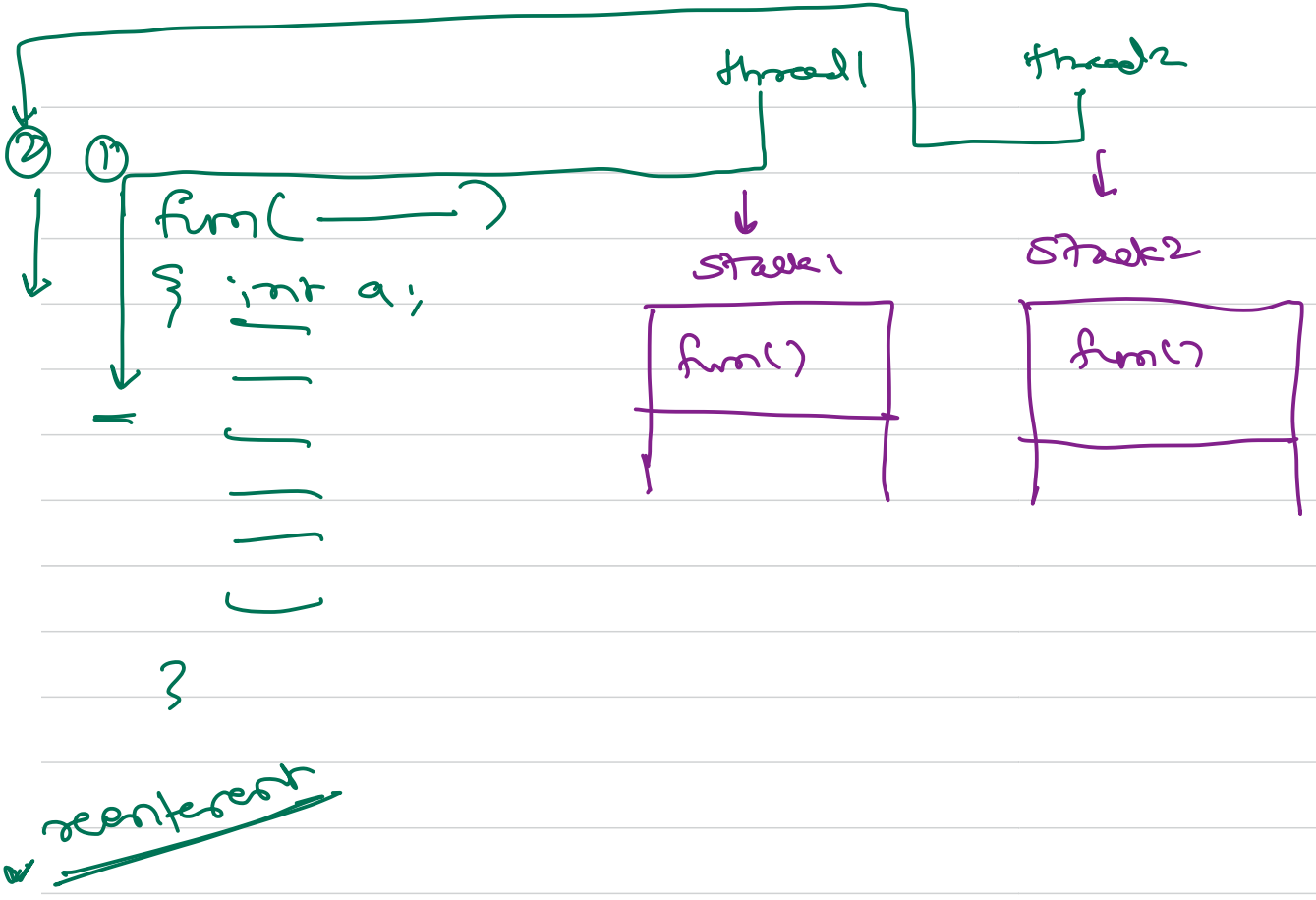


0	1	2	3	4	5	6	7	8
11	22	33	44	55	66	77	88	99

l > r

terminate/abort → error

stack overflow.

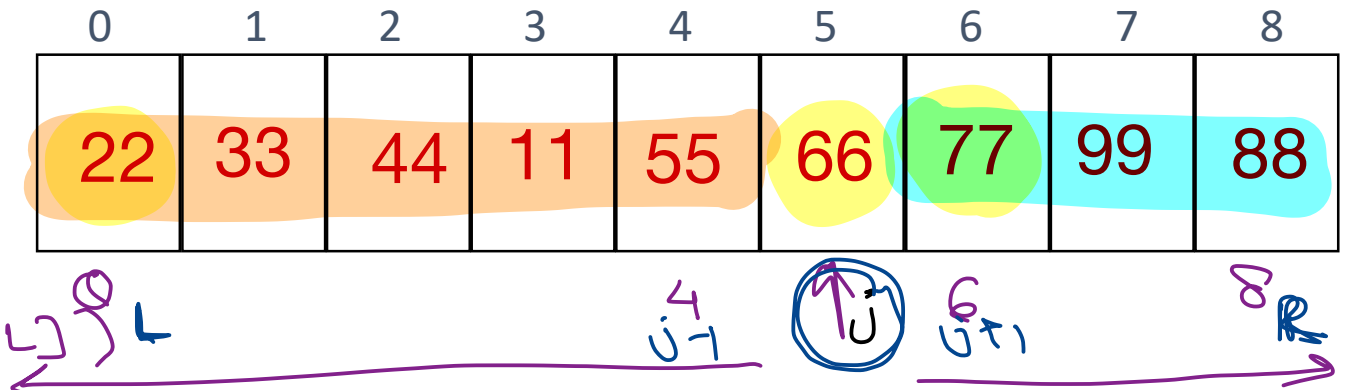

thread1
thread2
2
1
fun( )
{ int a;
}
Stack1
fun()
Stack2
fun()
reentrant

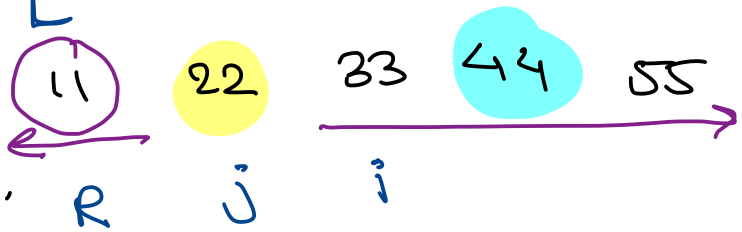
Quick Sort

i=L, j=R

terminating cond.→ L >= R

```
// Consider a[left] as pivot
// repeat till i & j not cross
while(i < j)
{
  // from left find ele > pivot
  while( i <= R && a[i] <= a[L] )
      i++;
  // from right find ele <= pivot
  while( a[j] > a[L] )
      j--;
  // if i & j not crossed, swap them.
  if(i < j)
      swap(a[i], a[j]);
}
// swap pivot with jth ele.
swap( a[L], a[j] );
```





77 88 99

```
quick_sort(arr, L, j-1);
quick_sort(arr, j+1, R);
```

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>