Data Structure & Algorithms

Trainer: Nilesh Ghule

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:

  $n! = 1 \times 2 \times 3 \times \dots \times n$

  - $n! = n * (n-1)!$ $\qquad 0! = 1$

  $x^y = x \times x \times x \times \dots$ y times

  - $x^y = X * x^{y-1}$ $\qquad \cancel{x^0 = 1}$
  - $T_n = T_{n-1} + T_{n-2}$ $\qquad T_1 = T_2 = 1$
  - 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);
```

120 $5! = 5 \times 4!$ 24

24 $4! = 4 \times 3!$ 6

6 $3! = 3 \times 2!$ 2

2 $2! = 2 \times 1!$ 1

1 $1! = 1 \times 0!$ 1

$0! = 1$ 1

FAR/Stack frame of function

1. arguments
2. local variables
3. return address.

Recursion

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

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

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

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

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

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

```
int main() {
    int res;
    res = fact(5);
    printf("%d", res);
    return 0;
}
```

$T \propto n$ → $O(n)$; n num of recursive fn calls.

Though time complexity is same as loop, still recursion is slower. Bcoz, fn call needs more time (due to FAR).

$S \propto n$ → recursion stack frames

$O(n)$ ← aux space or stack space

OS

stack
fact(0)
fact(1)
fact(2)
fact(3)
fact(4)
fact(5)
main()

Binary Search

- start : l = 0 & r = n-1.
- end : when l > r,
  stop → elem not found.
- find mid element
- compare with key
  & if matching return it
- if key < middle elem,
  find in left partition
  (left to mid-1)
- else // key > middle elem,
  find in right partition
  (mid+1 to right).

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

```
int binSearch(arr, key, left, right){
    if(left > right)
        return -1;
    mid = (left + right)/2;
    if(key == arr[mid])
        return mid;
    if (key < arr[mid])
        i = binSearch(arr, key, left, mid-1);
    else // key > arr[mid]
        i = binSearch(arr, key, mid+1, right);
    return i;
}
```

Selection Sort

5 6 3 8 4 2

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

0	1	2	3	4	5
5	6	3	8	4	2

i j

$itrs = (n-1) + (n-2) + (n-3) + \ldots + 1$

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

$T \propto \frac{n(n-1)}{2}$

$T \propto n^2 - n$

if $n >> 1$, $n^2 >>>> n$

then we can neglect lower order terms.

$T \propto n^2 \rightarrow \boxed{O(n^2)}$

itrs							
5	pass 1:	2 (i=0)	6 (j=1)	5	8	4	3 (j=5)
+ 4	pass 2:	2	3 (i=1)	6 (j=2)	8	5	4 (j=5)
+ 3	pass 3:	2	3	4 (i=2)	8 (j=3)	6	5 (j=5)
+ 2	pass 4:	2	3	4	5 (i=3)	8 (j=4)	6 (j=5)
+ 1	pass 5:	2	3	4	5	6 (i=4)	8 (j=5)
15							

Bubble Sort

5 6 3 8 4 2

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

0	1	2	3	4	5
2	3	4	5	6	8

j j+1

3

3

itrs $= (n-1) * (n-1)$

$= (n-1)^2$

$T \propto (n-1)^2$

$T \propto n^2 - 2n + 1$

$\boxed{T \propto n^2} \rightarrow O(n^2)$

itrs							
5	← pass 1:	5	3	6	4	2	8
5	← pass 2:	3	5	4	2	6	8
5	← pass 3:	3	4	2	5	6	8
5	← pass 4:	3	2	4	5	6	8
5	← pass 5:	2	3	4	5	6	8

further Improved Bubble Sort

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

0	1	2	3	4	5
2	3	4	5	6	8
				↑ j	↑ j1

Best Case: Array is already sorted.

only one pass needed

itrs = n-1

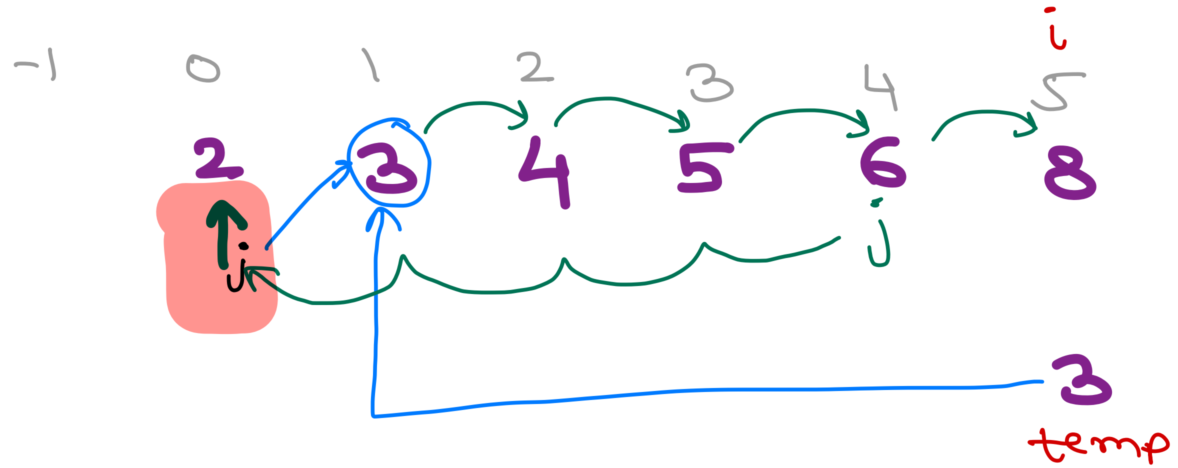
$T \propto n-1$

$T \propto n$

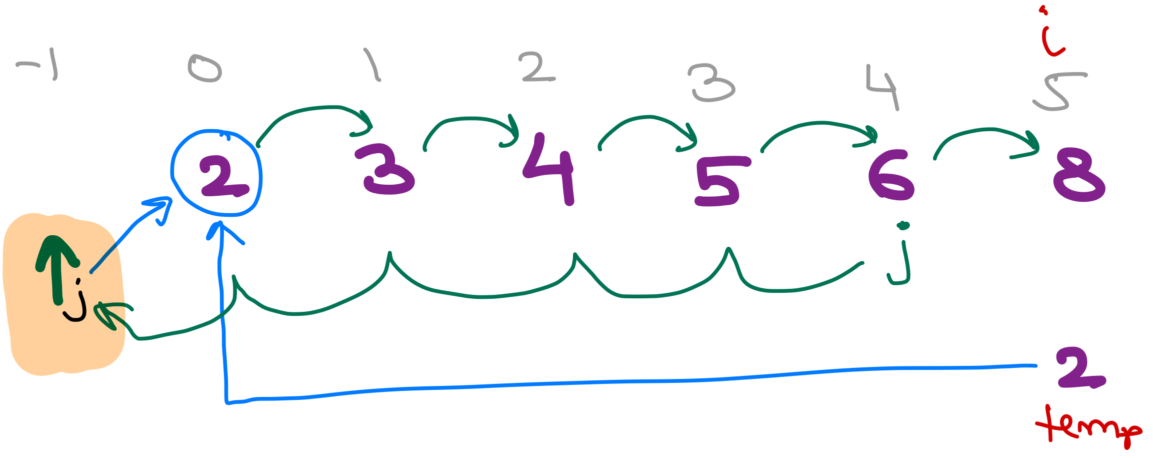
$\boxed{O(n)}$

Insertion ~~Sort~~ Technique

5 6 3 8 4 2



0	1	2	3	4	5




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

Insertion Sort

6 5 3 8 4 2

```
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;
}
```

0	1	2	3	4	5
6	5	3	8	4	2

i=1

5 6

i=2

3 5 6

i=3

3 5 6 8

i=4

3 4 5 6 8

i=5

2 3 4 5 6 8

Insertion Sort

0	1	2	3
4	3	2	1

① ← 3 4

② ← 2 3 4

③ ← 1 2 3 4

$$itrs = 1 + 2 + \ldots + (n-1)$$

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

$$T \propto \frac{n(n-1)}{2}$$

$$T \propto n^2 \rightarrow O(n^2)$$

Worst Case.

0	1	2	3	4	5

Avg Case $\rightarrow O(n^2)$

Insertion Sort

Best case

0	1	2	3
1	2	3	4

1 ← 1 2

1 ← 1 2 3

1 ← 1 2 3 4

itrs = (n-1)

$T \propto n-1$

$T \propto n$

$\boxed{O(n)}$

0	1	2	3	4	5

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>