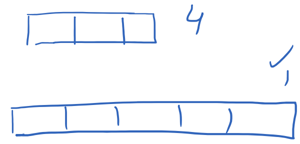
How to measure the performance of any algorithm?

Time ~~→~~ → Exact time | ~~Space~~ ??

Hardware Dependent

♀ → a+b 10 ms

S.C → a+b 0.1 ms



~~LOC~~

10

✓✓ Operations / Calculations

→ Optimal number of operations ✓

To measure the number of operations, performed by any algorithm ⇒ Asymptotic Analysis

---

Arrays ⇒ Linear Search

→ How many times do you visit every element? (1)

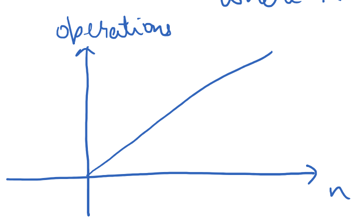
n elements

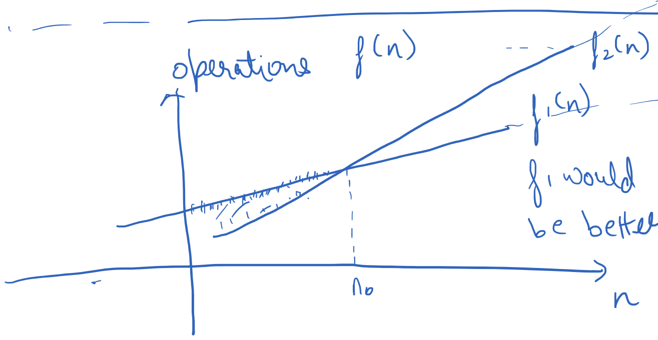
10 , 10000

$f(n) = \underbrace{K \cdot n}$

input

→ no. of operations where K is a constant





f1 would be better

n → lower, higher

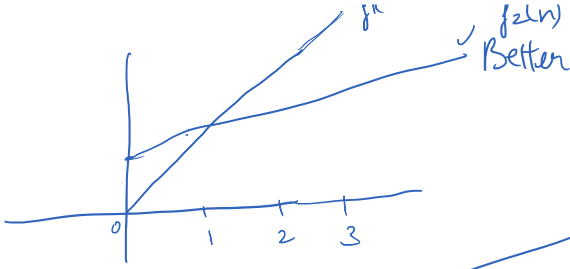
lower: 100

higher: 1000000

On comparison, higher values of n will have more impact.

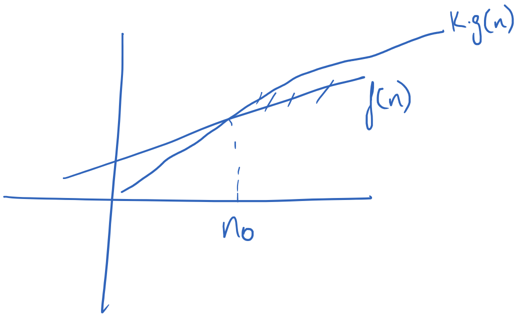
$f\ n) = 3 \cdot n$

$f_2(n) = n + \underline{3}$



Constants
↳ Ignore ??
↳ Big "O" Notation

↳ Worst case mein for large values of n, hamari algo f(n) ko approximate kro to we obtain g(n) such that



$$K \cdot g(n) \geq f(n) \quad \forall\, n \geq n_0$$

$$O(f(n)) = g(n)$$

$$f(n) = a_0 n^x + a_1 n^{x-1} + a_2 n^{x-2} + \ldots + a_x n^0$$

$$f'(n) = a_0 n^x + a_1 n^x + a_2 n^x + \ldots + a_x n^x$$

$$= n^x (a_0 + a_1 + a_2 + \ldots + a_x)$$

$$= n^x \cdot K$$

$$= \underbrace{K \cdot \boxed{n^x}}_{g(n)}$$

$$f(n) \leq K \cdot g(n)$$

$$O(f(n)) = g(n)$$

$$= n^x$$

$x = 3$

$n = 10$

$a_1 \times 10^2$

$a_1 \times 10^3$

0.1
3.1
4
K.1

Polynomial : O(Highest Power)

→ Mathematical operations: $a+b \rightarrow O(1)$

→ Variables : int a = 5; } → O(1)
int[] arr = new int [10];

↳ indexing } → O(1)
arr[0] = 10

① int n = Scn.nextInt(); → K

```
i=0; → K
while(i<n){        (K)
  dy80(i); → K
  i++; → K
}
```

$f(n) = K + K + K\cdot n + K\cdot n + K\cdot n$

$f(n) = 3K\cdot n + 2K$

$f(n) = K_0 . n + K_1$

$\Rightarrow O(n)$

(2)

```
int n= scn.nextInt(); → K
int i=0; → K
while (i≤n){        (K)
  i+=2; → K
}
```

$f(n) = 2K + K\cdot\frac{n}{2} + K\cdot\frac{n}{2}$

$= 2K + Kn$

$= K_0 . n + K_1$

$\Rightarrow O(n)$

(3)

```
int n ; → K
int i= 0; → K
while (i<n){        (K)
  i+=3; → K
  i+=2; → K
}
```

$f(n) = 2K + \frac{n}{5} \times 3K$

$= K_1 + n . K_0$

$\Rightarrow O(n)$

(4)

```
int n;
int i=0;
while(i≤n){
  i+=c;
}
```

$f(n) = 2K + 2\times\frac{n}{c}\times K$

$= K_1 + n . K_0$

$\Rightarrow O(n)$

loop	i
1	0
2	c
3	2c
⋮	3c
	4c
	⋮
	$i = (l-1)c$

$(l-1)c = n$

$l = \frac{n}{c} + 1$

```
int n ;
int i=1;
while(i≤n){
  i *=2;
}
```

loop	i
1	1

```
    i *= 2;
}
```

$$f(n) = K + K + 2K \times \log_2 n$$
$$= 2K + 2K \log_2 n$$
$$= K' + K^0 \log n$$

$\Rightarrow O(\log n)$

loop	i
1	1
2	2
3	4
⋮	
l	$2^{l-1} = n$

$2^{l-1} \approx n$

$2^{l-1} = n$

$\log_2 2^{l-1} = \log_2 n$

$(l-1) \cancel{\log_2 2} = \log_2 n$

$l = \log_2 n + 1$

II)

```
int n;
int i=1;
while (i <= n){
    i *= 3;
}
```

$\Rightarrow O(\log n)$

III)

```
int n;
int i=1;
while(i<=n){
    i*=K
}
```

$\Rightarrow$

$\Rightarrow O(\log n)$

loop	i
1	1
2	K
3	K^2
⋮	⋮
l	$n = K^{l-1}$

$l = \log_K n + 1$

IV)

```
int n;
while( n>0){
    n--;
}
```

$\Rightarrow O(n)$

```
int n;
while (n>=0){
    n = n-c;
}
```

$\Rightarrow O(n)$

V)

```
n;
while (n>=0){
    n = n/K;
}
```

loop	i
1	n
2	n/K
3	n/K^2

}

↳ $O(\log n)$

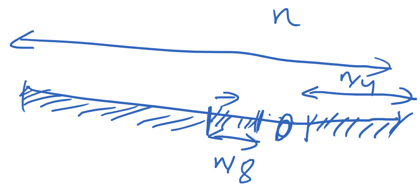
3	n/K^2
$\vdots$	$\vdots$
l	$1 \Rightarrow n/K^{l-1}$

$$\frac{n}{K^{l-1}} = 1$$

$$n = K^{l-1}$$

$$\log n = (l-1) \log K$$

$$\log_K n + 1 = l$$



1) Linear Search

↳ $O(n)$

2) Binary Search

↳ $O(\log n)$

loop	elements
1	n
2	$n/2$
3	$n/4$
$\vdots$	
l	$n/2^{l-1} = 1$

$$n = 2^{l-1}$$

$$\log_2 n + 1 = l$$

①
```
for (int i=0; i*i < n; i++){
    Syso(Hi);
}
```
↳ $O(\sqrt{n})$

100	i
1	0
2	1
3	2
4	3
ℓ	$(\ell-1)$

$i*i = n$

$(\ell-1)(\ell-1) = n$

$(\ell-1) = \sqrt{n}$

$\ell = \sqrt{n} + 1$

2)
```
for (int i= 1; i<=n; i++) {
    for(int j=1; j<=i; j++){
        for(int k=1; k<=1000; k++){
            // Code
        }
    }
}
```
↳ $O(n^2)$

i=1, j=1, k=1000 | =1000 | i=3: j=1, j=2, j=3 | . . | i=n: j=1, j=2, …, j=n

1×1000 | 1×1000, 1+1000 | 1×1000, 1×1000, 1×1000 | …. | 1×1000, 1×1000, …, n times

$1\times1000 + 2\times1000 + 3\times1000 + \ldots + n\times1000$

$1000(1+2+3+\ldots+n)$

$1000 \times \frac{(n)(n+1)}{2} \Rightarrow (n^2+n)\times 500$

3)
```
for( i=1; i<=n ; i++) {
    for(j=1; j<=i^2 ; j++) {
        for(k=1; k<=n/2; k++) {
            // Code
        }
    }
}
```
↳ $O(n^4)$

i=1: j=1, $k=\frac{n}{2}$ | i=2: $j=1, k\to\frac{n}{2}$; $j=2\to k\to\frac{n}{2}$; $j=3, k\to\frac{n}{2}$; $j=4, k\to\frac{n}{2}$ | …… | i=n: $j=1, k\to\frac{n}{2}$; $j=2, k\to\frac{n}{2}$; …; $j=n^2\to k\to\frac{n}{2}$

$\frac{n}{2} + 2^2\times\frac{n}{2} + 3^2\times\frac{n}{2} + \ldots n^2\times\frac{n}{2}$

$\frac{n}{2}(1+2^2+3^2+\ldots+n^2)$

$\frac{n}{2}\left(\frac{n(n+1)(2n+1)}{6}\right)$

$O(n^4)$
↳ Time Complexity

Space Complexity

$O(n^4) = k_0 n^4 + K_1 n^3 + K_2 n^2 + K_3 n^1 + K_4 n^0$

H.W

→ Bubble Sort
→ Selection Sort
→ Insertion Sort

---

3) for(int i=1 ; i ≤ n ; i*=2){

}

↳ $O(\log n)$

loop	i
1	1
2	2
3	4
⋮	⋮
l	$i = 2^{l-1} = n$

$l = \log_2 n + 1$

---

5) for(int i=n/2 ; i ≤ n; i++){
 for(int j=1; j ≤ n/2; j++){
  for(int k=1; k ≤ n ; k*=2){
  }
 }
}

$i = n/2$
$j = 1, k \to \log n$
$j = 2, k \to \log n$
$j = 3, k \to \log n$
$j = \frac{n}{2}, k \to \log n$

$i = n/2$ $\quad$ $i = n/2+1$ $\quad \dots \quad$ $i = n$

$\log n \times \frac{n}{2} + \log n \times \frac{n}{2} + \dots \log n \times \frac{n}{2}$

$= \frac{n}{2} \times \left(\log n \times \frac{n}{2}\right)$

$= n^2 \log n / 4$

6) for(i=$\frac{n}{2}$; i ≤ n; i++){
 for(j=1; j ≤ n; j*=2){
  for(k=1; k<n ; k*=2){
  }
 }
}

$i = n/2$
$j = 1 \quad k \to \log n$
$j = 2 \quad k \to \log n$
$j = 4 \quad k \to \log n$
⋮
$j = \log n \quad k \to \log n$

$i = \frac{n}{2}+1$
} (for C cthr)

$\log n \times \log n + \log n \times \log n + \dots \frac{n}{2}$ times

$\frac{n}{2} (\log n)^2$

$\Rightarrow O(n(\log n)^2)$

7) for(int i=1 ; i ≤ n ; i++){

$i=1 \quad i=2 \quad i=3$

```
    for(j=1; j<=n; j++){
    }
}
```

$\Rightarrow n + n + n \ldots n$ times

$\Rightarrow n^2$

$\hookrightarrow O(n^2)$

8)
```
for( i=1 ; i<=n; i+=k){
    for(j=1 ; j<=k ; j++){
    }
}
```

$\hookrightarrow O(n)$

$i=1$ $i=k$ $i=2k$

$j \to 1, 2, 3, \ldots, k$ $j \to 1, 2, 3, \ldots, k$

$k + k + k + \ldots \left(\frac{n}{k}+1 \text{ times}\right)$

$\cancel{k} \times \left(\frac{n}{\cancel{k}}+1\right)$ times

$O(n+k) \Rightarrow O(n)$

```
int [] arr= new int[n];
  for(i=1; i<n ;i++){
      for (j=0 ; j< 10000 ; j++){
      }
  }
```

$\to$ Constant

$\hookrightarrow O(n)$

Order of Complexities

$1 < \log(\log n) < \sqrt{\log n} < \log n < n < n.\log n < n^2 < n^2 \log n < n^3 < c^n < n! < n^n < n^{n^n}$