Ordo - big-O

Johan Montelius

KTH

HT23

big-O

An estimate of the change in execution time...
when the data set grows large.

big-O

An estimate of the change in execution time...

when the data set grows large.

big-O

An estimate of the change in execution time...
when the data set grows large.

an example

```
public static boolean search(int[] arr, int key) {
  for(int i = 0; i < arr.length; i++) {
    if (arr[i] == key )
    return true;
  }
  return false;
}
```

- c1: set up arguments
- c2: $i = 0$
- c3: $i < \text{arr.length}$
- c4: $\text{arr}[i] == \text{key}$
- c5: $i{+}{+}$
- c6: return

an example

```
public static boolean search(int[] arr, int key) {
  for(int i = 0; i < arr.length; i++) {
    if (arr[i] == key )
    return true;
  }
  return false;
}
```

- c1: set up arguments
- c2: $i = 0$
- c3: $i <$ arr.length
- c4: arr[i] $==$ key
- c5: i$++$
- c6: return

an example

```
public static boolean search(int[] arr, int key) {
  for(int i = 0; i < arr.length; i++) {
    if (arr[i] == key )
    return true;
  }
  return false;
}
```

- c1: set up arguments
- c2: $i = 0$
- c3: $i <$ arr.length
- c4: arr[i] $==$ key
- c5: i$++$
- c6: return

an example

```
public static boolean search(int[] arr, int key) {
  for(int i = 0; i < arr.length; i++) {
    if (arr[i] == key )
    return true;
  }
  return false;
}
```

- c1: set up arguments
- c2: $i = 0$
- c3: $i <$ arr.length
- c4: $\text{arr}[i] == \text{key}$
- c5: $i{+}{+}$
- c6: return

an example

```
public static boolean search(int[] arr, int key) {
  for(int i = 0; i < arr.length; i++) {
    if (arr[i] == key )
    return true;
  }
  return false;
}
```

- c1: set up arguments
- c2: $i = 0$
- c3: $i <$ arr.length
- c4: $\text{arr[i]} == \text{key}$
- c5: $i{+}{+}$
- c6: return

an example

```
public static boolean search(int[] arr, int key) {
  for(int i = 0; i < arr.length; i++) {
    if (arr[i] == key )
    return true;
  }
  return false;
}
```

- c1: set up arguments
- c2: i = 0
- c3: i < arr.length
- c4: arr[i] == key
- c5: i++
- c6: return

an example

```
public static boolean search(int[] arr, int key) {
  for(int i = 0; i < arr.length; i++) {
    if (arr[i] == key )
    return true;
  }
  return false;
}
```

- c1: set up arguments
- c2: i $=$ 0
- c3: i $<$ arr.length
- c4: arr[i] $==$ key
- c5: i$++$
- c6: return

an example

```
public static boolean search(int[] arr, int key) {
  for(int i = 0; i < arr.length; i++) {
    if (arr[i] == key )
    return true;
  }
  return false;
}
```

- c1: set up arguments
- c2: $i = 0$
- c3: $i <$ arr.length
- c4: arr[i] $==$ key
- c5: $i{+}{+}$
- c6: return

the execution time

$$t(n) = c_1 + c_2 + (c_3 + c_4 + c_5) \times n + c_3 + c_6$$

$$c_7 = c_3 + c_4 + c_5$$

$$c_8 = c_1 + c_2 + c_3 + c_6$$

$$t(n) = c_7 \times n + c_8$$

the execution time

$$t(n) = c_1 + c_2 + (c_3 + c_4 + c_5) \times n + c_3 + c_6$$

$$c_7 = c_3 + c_4 + c_5$$

$$c_8 = c_1 + c_2 + c_3 + c_6$$

$$t(n) = c_7 \times n + c_8$$

the execution time

$$t(n) = c_1 + c_2 + (c_3 + c_4 + c_5) \times n + c_3 + c_6$$

$$c_7 = c_3 + c_4 + c_5$$

$$c_8 = c_1 + c_2 + c_3 + c_6$$

$$t(n) = c_7 \times n + c_8$$

the execution time

$$t(n) = c_1 + c_2 + (c_3 + c_4 + c_5) \times n + c_3 + c_6$$

$$c_7 = c_3 + c_4 + c_5$$

$$c_8 = c_1 + c_2 + c_3 + c_6$$

$$t(n) = c_7 \times n + c_8$$

big-O

$$t(n) = c_7 \times n + c_8$$

$$t(n) \in O(n)$$

since ....
... there is a k such that $k \times n$ is always greater than $t(n)$ from some (large) value of n

big-O

$$t(n) = c_7 \times n + c_8$$

$$t(n) \in O(n)$$

since ....
... there is a k such that $k \times n$ is always greater than $t(n)$ from some (large) value of n

big-O

$$t(n) = c_7 \times n + c_8$$

$$t(n) \in O(n)$$

since ....

... there is a k such that $k \times n$ is always greater than $t(n)$ from some (large) value of n

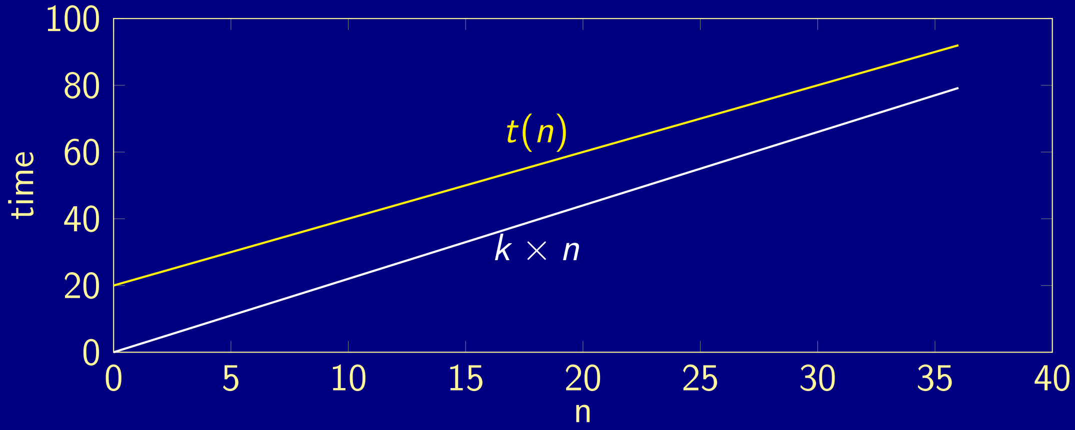
big-O

$$t(n) = c_7 \times n + c_8$$

$$t(n) \in O(n)$$

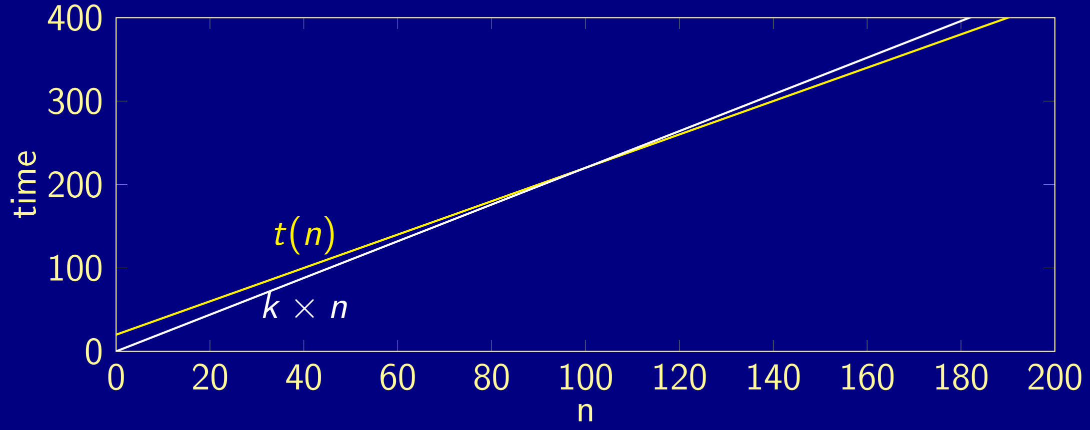
since ....
... there is a k such that $k \times n$ is always greater than $t(n)$ from some (large) value of n

nota bene



$c_7 = 2,\ c_8 = 20\ \ k = 2.2$

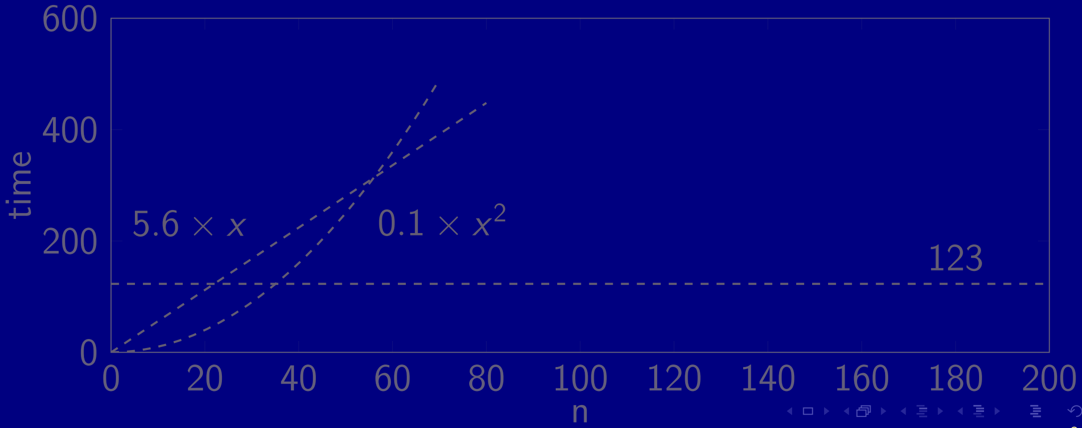
nota bene



400
300
200
100
0
time
$t(n)$
$k \times n$
0
20
40
60
80
100
120
140
160
180
200
n

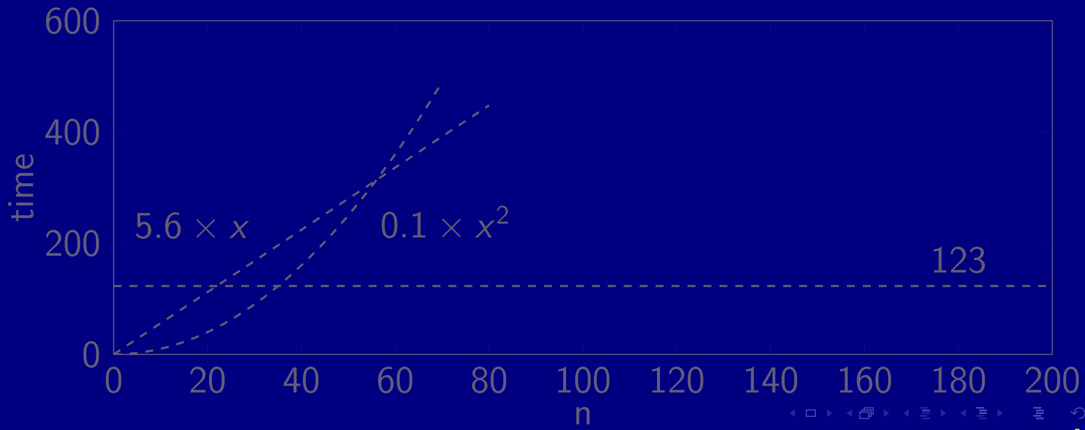
What about this?

$$t(n) = 0.1 \times n^2 + 5.6 \times n + 123$$



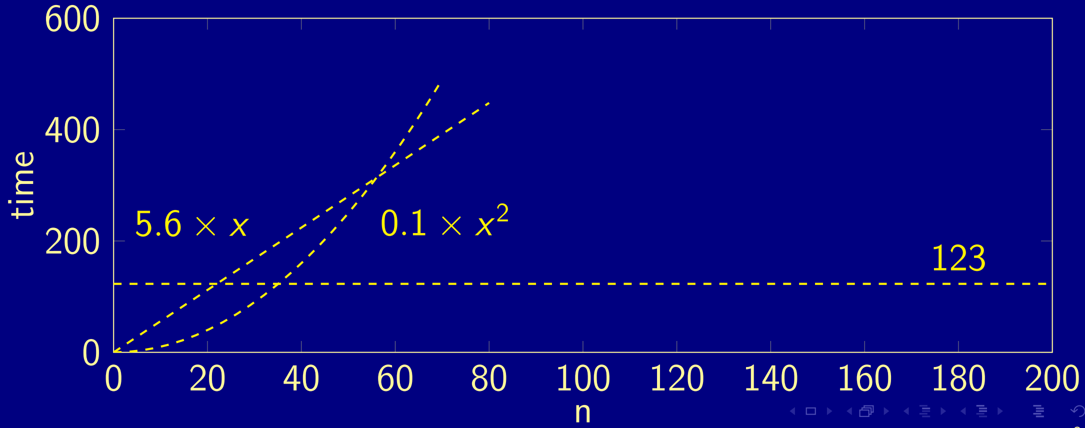
What about this?

$$t(n) = 0.1 \times n^2 + 5.6 \times n + 123$$



What about this?

$$t(n) = 0.1 \times n^2 + 5.6 \times n + 123$$



kuggfråga

$$t(n) \in O(n^2)$$

What is the execution time for $n = 1000$?
What will happen if we double the size of a large data set?

kuggfråga

$$t(n) \in O(n^2)$$

What is the execution time for $n = 1000$?
What will happen if we double the size of a large data set?

kuggfråga

$$t(n) \in O(n^2)$$

What is the execution time for $n = 1000$?

What will happen if we double the size of a large data set?

kuggfråga

$$t(n) \in O(n^2)$$

What is the execution time for $n = 1000$?
What will happen if we double the size of a large data set?

kuggfråga

$$t(n) = 0.1 \times n^2 + 5.6 \times n + 123$$

$$t(10) = 189ns$$

$$t(1000) = 105\mu s$$

$$t(20) = 275ns$$

$$t(2000) = 411\mu s$$

$$t(40) = 507ns$$

$$t(4000) = 1620\mu ns$$

$$t(80) = 1211ns$$

$$t(8000) = 6440\mu s$$

kuggfråga

$$t(n) = 0.1 \times n^2 + 5.6 \times n + 123$$

$t(10) = 189ns$

$t(1000) = 105\mu s$

$t(20) = 275ns$

$t(2000) = 411\mu s$

$t(40) = 507ns$

$t(4000) = 1620\mu ns$

$t(80) = 1211ns$

$t(8000) = 6440\mu s$

kuggfråga

$$t(n) = 0.1 \times n^2 + 5.6 \times n + 123$$

$t(10) = 189ns$

$t(1000) = 105\mu s$

$t(20) = 275ns$

$t(2000) = 411\mu s$

$t(40) = 507ns$

$t(4000) = 1620\mu ns$

$t(80) = 1211ns$

$t(8000) = 6440\mu s$

kuggfråga

$$t(n) = 0.1 \times n^2 + 5.6 \times n + 123$$

$t(10) = 189ns$

$t(1000) = 105\mu s$

$t(20) = 275ns$

$t(2000) = 411\mu s$

$t(40) = 507ns$

$t(4000) = 1620\mu ns$

$t(80) = 1211ns$

$t(8000) = 6440\mu s$

kuggfråga

$$t(n) = 0.1 \times n^2 + 5.6 \times n + 123$$

$t(10) = 189 ns$

$t(1000) = 105 \mu s$

$t(20) = 275 ns$

$t(2000) = 411 \mu s$

$t(40) = 507 ns$

$t(4000) = 1620 \mu ns$

$t(80) = 1211 ns$

$t(8000) = 6440 \mu s$

kuggfråga

$$t(n) = 0.1 \times n^2 + 5.6 \times n + 123$$

$$t(10) = 189ns \qquad t(1000) = 105\mu s$$

$$t(20) = 275ns \qquad t(2000) = 411\mu s$$

$$t(40) = 507ns \qquad t(4000) = 1620\mu ns$$

$$t(80) = 1211ns \qquad t(8000) = 6440\mu s$$

kuggfråga

$$t(n) = 0.1 \times n^2 + 5.6 \times n + 123$$

$t(10) = 189ns$

$t(20) = 275ns$

$t(40) = 507ns$

$t(80) = 1211ns$

$t(1000) = 105\mu s$

$t(2000) = 411\mu s$

$t(4000) = 1620\mu ns$

$t(8000) = 6440\mu s$

kuggfråga

$$t(n) = 0.1 \times n^2 + 5.6 \times n + 123$$

$t(10) = 189ns$

$t(20) = 275ns$

$t(40) = 507ns$

$t(80) = 1211ns$

$t(1000) = 105\mu s$

$t(2000) = 411\mu s$

$t(4000) = 1620\mu ns$

$t(8000) = 6440\mu s$

kuggfråga

$$t(n) = 0.1 \times n^2 + 5.6 \times n + 123$$

$$t(10) = 189ns$$

$$t(1000) = 105\mu s$$

$$t(20) = 275ns$$

$$t(2000) = 411\mu s$$

$$t(40) = 507ns$$

$$t(4000) = 1620\mu ns$$

$$t(80) = 1211ns$$

$$t(8000) = 6440\mu s$$

kuggfråga

$$t(n) = 0.1 \times n^2 + 5.6 \times n + 123$$

$$t(10) = 189 ns \qquad t(1000) = 105 \mu s$$

$$t(20) = 275 ns \qquad t(2000) = 411 \mu s$$

$$t(40) = 507 ns \qquad t(4000) = 1620 \mu ns$$

$$t(80) = 1211 ns \qquad t(8000) = 6440 \mu s$$

log, lg, ln ..

$$log_a(x) = \frac{log_b(x)}{log_b(a)}$$

$$log_{10}(n) = \frac{log_2(n)}{log_2(10)}$$

$$log_{10}(n) = k \times log_2(n)$$

$$k \times log_2(n) = O(log_2(n))$$

log, lg, ln ..

$$log_a(x) = \frac{log_b(x)}{log_b(a)}$$

$$log_{10}(n) = \frac{log_2(n)}{log_2(10)}$$

$$log_{10}(n) = k \times log_2(n)$$

$$k \times log_2(n) = O(log_2(n))$$

log, lg, ln ..

$$log_a(x) = \frac{log_b(x)}{log_b(a)}$$

$$log_{10}(n) = \frac{log_2(n)}{log_2(10)}$$

$$log_{10}(n) = k \times log_2(n)$$

$$k \times log_2(n) = O(log_2(n))$$

log, lg, ln ..

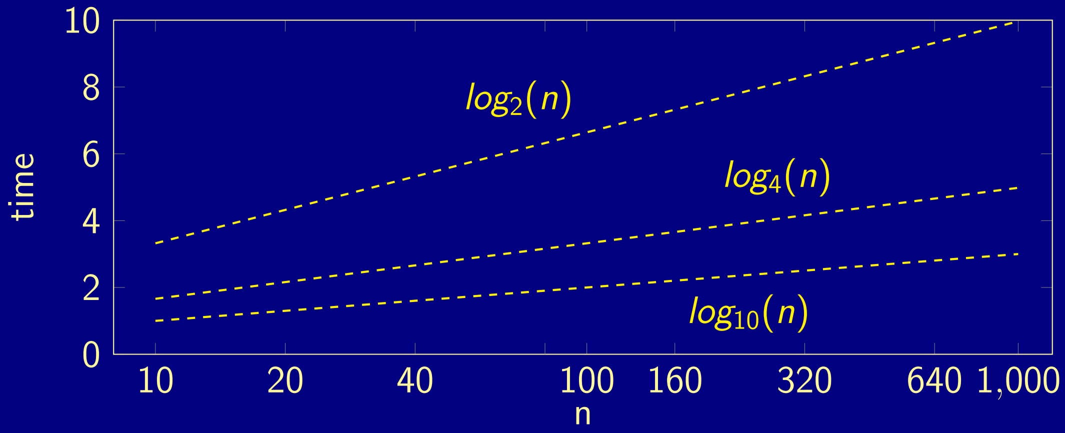
$$log_a(x) = \frac{log_b(x)}{log_b(a)}$$

$$log_{10}(n) = \frac{log_2(n)}{log_2(10)}$$

$$log_{10}(n) = k \times log_2(n)$$

$$k \times log_2(n) = O(log_2(n))$$

which log scale



best, worst or average

Linear search in an array of size n.

- If you're lucky, you will find it in the first position - $O(1)$
- If you're not lucky, you will have to search to the end - $O(n)$
- In average you will have to search through half the array - $O(n)$

We often only care about the average case - but need to be aware of the worst case.

best, worst or average

Linear search in an array of size n.

- If you're lucky, you will find it in the first position - $O(1)$
- If you're not lucky, you will have to search to the end - $O(n)$
- In average you will have to search through half the array - $O(n)$

We often only care about the average case - but need to be aware of the worst case.

best, worst or average

Linear search in an array of size n.

- If you're lucky, you will find it in the first position - $O(1)$
- If you're not lucky, you will have to search to the end - $O(n)$
- In average you will have to search through half the array - $O(n)$

We often only care about the average case - but need to be aware of the worst case.

best, worst or average

push() operation in a dynamic stack

- If the stack is big enogh - $O(1)$
- If you have to increase the stack - $O(n)$
- In average ..... ?

best, worst or average

push() operation in a dynamic stack

- If the stack is big enogh - $O(1)$
- If you have to increase the stack - $O(n)$
- In average ..... ?

best, worst or average

push() operation in a dynamic stack

- If the stack is big enogh - $O(1)$
- If you have to increase the stack - $O(n)$
- In average ..... ?

best, worst or average

push() operation in a dynamic stack

- If the stack is big enogh - $O(1)$
- If you have to increase the stack - $O(n)$
- In average ..... ?

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

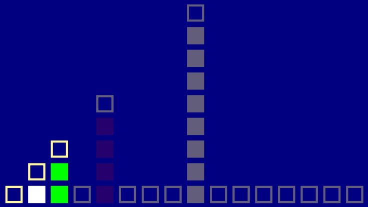
best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack



Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

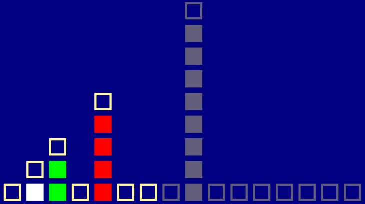
best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack



Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....

best, worst or average

push() operation in a dynamic stack

Amortized cost of push() operation is ....