Mobile Team Training: Session 2

Asymptotic Notations, Dynamic and Static Arrays

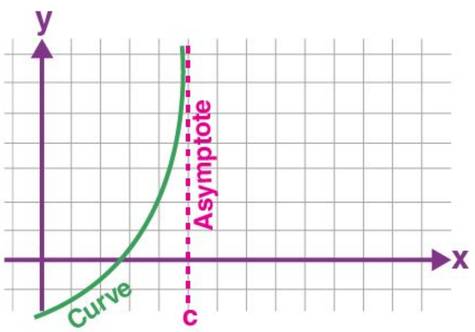
By Sardorbek Abdulabbozov

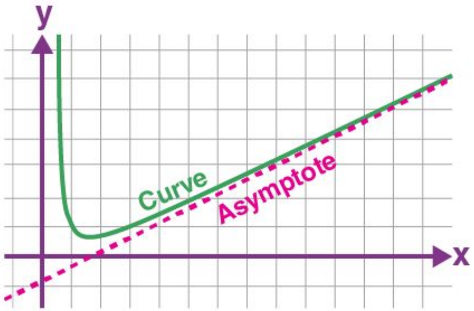
Part I:

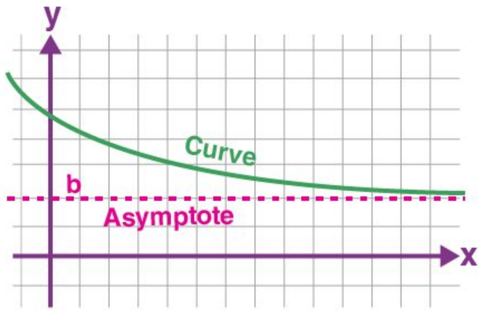
Asymptotic Notations

What is asymptote?

Asymptote is a line that a curve approaches as it moves towards infinity.





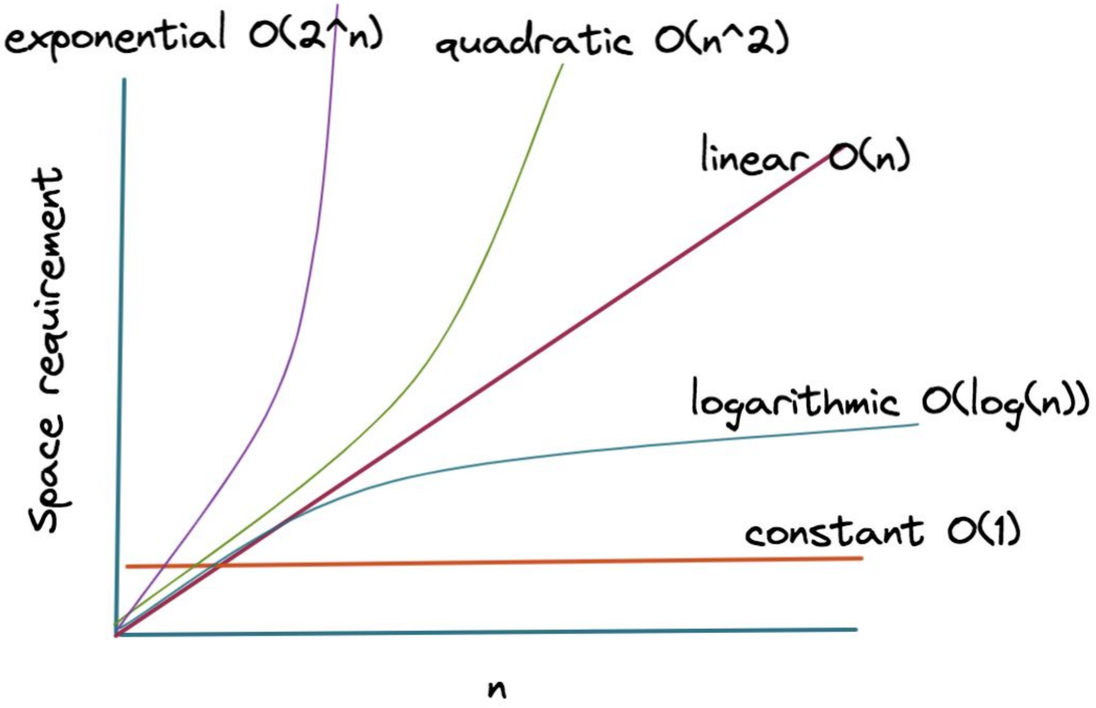


Asymptotic Notations

Asymptotic Notation is used to describe the running time of an algorithm - how much time an algorithm takes with a given input, n. There are three different notations: O - the worst-case, Theta (Θ) - the average-case, and Omega (Ω) - the best-case.

Each of them has two different types:
O: Big O & Small O
Theta (Θ): Big Θ & Small Θ
Omega (Ω): Big Ω & Small Ω

Time Complexity

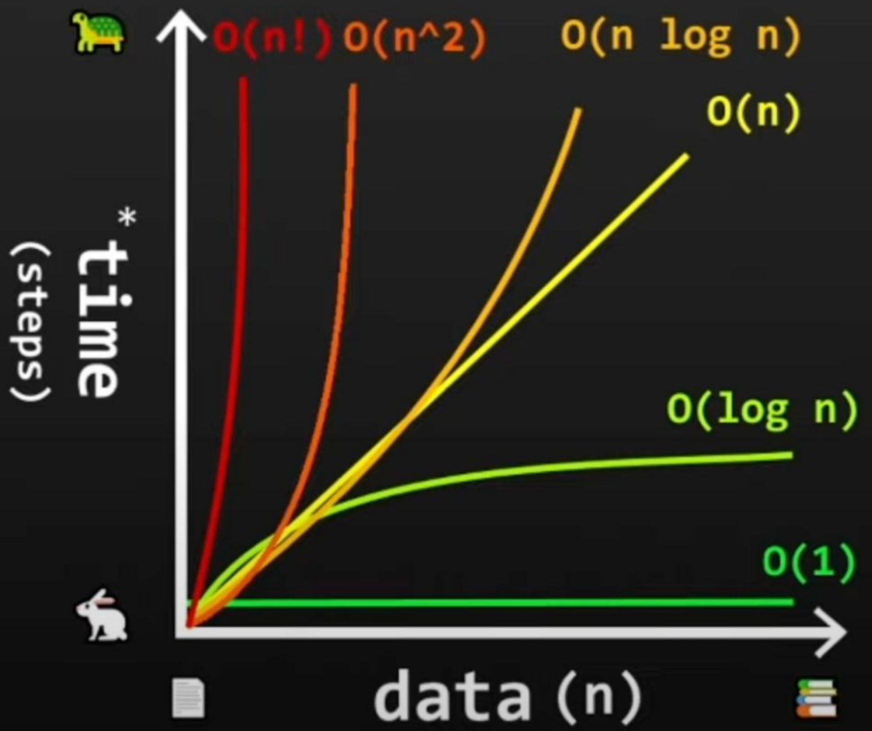


The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input

Big O (or simply O(n))

"How code slows as data grows."

Big O notation



O(1) = constant time
- random access of an element in an array
- inserting at the beginning of linkedlist

O(log n) = logarithmic time
- binary search

O(n) = linear time
- looping through elements in an array
- searching through a linkedlist

O(n log n) = quasilinear time
- quicksort
- mergesort
- heapsort

O(n^2) = quadratic time
- insertion sort
- selection sort
- bubblesort

O(n!) = factorial time
- Traveling Salesman Problem

Big-O Properties

$$O(n + c) = O(n)$$

$$O(cn) = O(n), \; c > 0$$

Let f be a function that describes the running time of a particular algorithm for an input of size n:

$$f(n) = 7\log(n)^3 + 15n^2 + 2n^3 + 8$$

$$O(f(n)) = O(n^3)$$

Big-O Examples

The following run in constant time: **O(1)**

```
a := 1
b := 2
c := a + 5*b
```

```
i := 0
While i < 11 Do
    i = i + 1
```

Big-O Examples

The following run in linear time: **O(n)**

```
i := 0
While i < n Do
    i = i + 1
```

$f(n) = n$
$O(f(n)) = \mathbf{O(n)}$

```
i := 0
While i < n Do
    i = i + 3
```

$f(n) = n/3$
$O(f(n)) = \mathbf{O(n)}$

Big-O Examples

Both of the following run in quadratic time. The first may be obvious since n work done n times is $n*n$ = **O(n²)**, but what about the second one?

```
For (i := 0 ; i < n; i = i + 1)
    For (j := 0 ; j < n; j = j + 1)
```

$$f(n) = n*n = n^2,\ O(f(n)) = O(n^2)$$

```
For (i := 0 ; i < n; i = i + 1)
    For (j := i ; j < n; j = j + 1)
              ^ replaced 0 with i
```

Big-O Examples

```
i := 0
While i < n Do
    j = 0
    While j < 3*n Do
        j = j + 1
    j = 0
    While j < 2*n Do
        j = j + 1
    i = i + 1
```

$$f(n) = n * (3n + 2n) = 5n^2$$

$$O(f(n)) = \mathbf{O(n^2)}$$

Big-O Examples

```
i := 0
While i < 3 * n Do
    j := 10
    While j <= 50 Do
        j = j + 1
    j = 0
    While j < n*n*n Do
        j = j + 2
    i = i + 1
```

$$f(n) = 3n * (40 + n^3/2) = 3n/40 + 3n^4/2$$

$$O(f(n)) = \mathbf{O(n^4)}$$

Let's practice

Write a program that calculates the sum of first 10,000 integers

```
int addUp(int n) {
  int sum = 0;
  for (int i = 1; i <= n; i++) {
    sum += i;
  }
  return sum;
}
```

O(n) - linear time complexity

Is this the most efficient way to solve the problem?

```
int addUp(int n) {
  int sum = (n * (n + 1)) ~/ 2;
  return sum;
}
```

This is the most efficient solution to the given problem. Why?

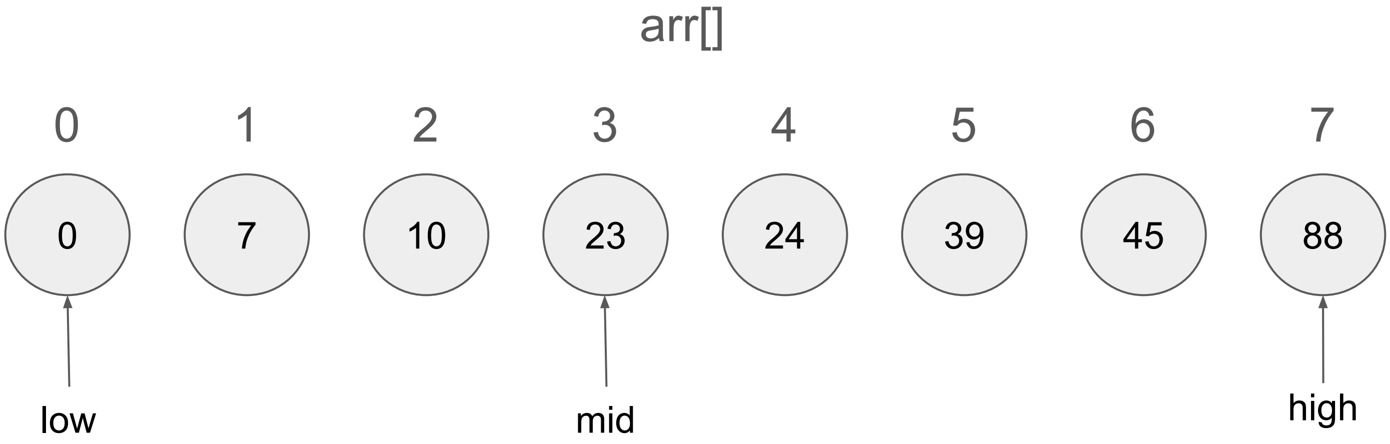
Because, time complexity is constant: O(1)

Let’s have an example to understand Big O better from DSA perspective...

Binary Search Algorithm is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to $O(\log N)$.

In following example, we’ll observe one of the worst case scenarios of binary research - case when search value is the last element of the list.

Binary Search



n = 8
value = 88

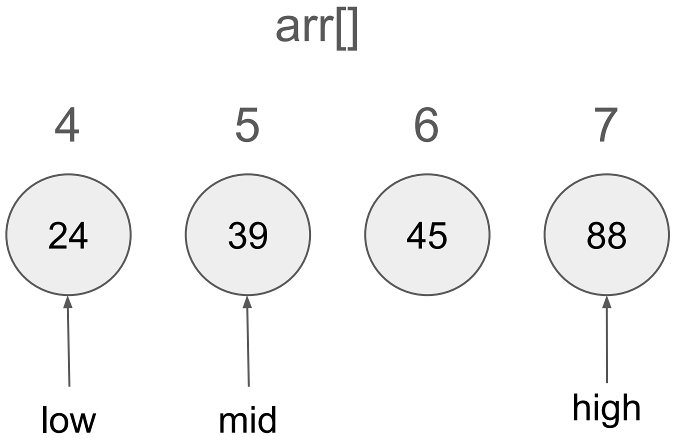
low = 0
high = 7
mid = floor((low + high) / 2) = 3

if (arr[mid] == value) return mid;

else if (arr[mid] < value) low = mid + 1;

else if (arr[mid] > value) high = mid - 1;

Binary Search



n = 8
value = 88

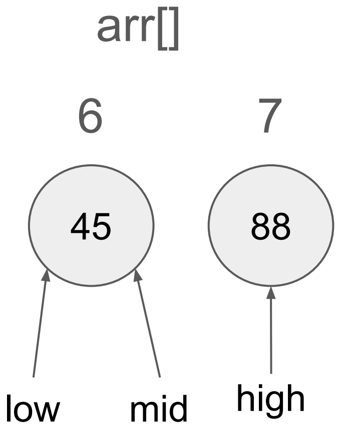
low = 4
high = 7
mid = floor((low + high) / 2) = 5

if (arr[mid] == value) return mid;

else if (arr[mid] < value) low = mid + 1;

else if (arr[mid] > value) high = mid - 1;

Binary Search



n = 8
value = 88

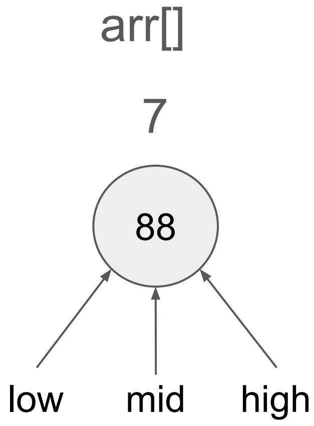
low = 6
high = 7
mid = floor((low + high) / 2) = 6

```
if (arr[mid] == value) return mid;

else if (arr[mid] < value) low = mid + 1;

else if (arr[mid] > value) high = mid - 1;
```

Binary Search



$O(n) = \log(n)$*

$O(8) = \log(8) = 3$

* $O(n) = \log(n)$ means number of times that array is divided by half to perform search operation in worst-case

n = 8
value = 88 (VALUE FOUND)

low = 7
high = 7
mid = floor((low + high) / 2) = 7

if (arr[mid] == value) return mid;

else if (arr[mid] < value) low = mid + 1;

else if (arr[mid] > value) high = mid - 1;

Implementation in Dart

```dart
void main() {
  final List<int> arr = [0, 7, 10, 23, 24, 39, 45, 88];

  final int searchValue = 88;

  final index = binarySearch(searchValue, arr);

  print(index != -1 ? "$searchValue found at index $index." : "Value Not Found");
}

int binarySearch(int searchValue, List<int> arr) {
  int low = 0;
  int high = arr.length - 1;
  late int mid;

  while (low <= high) {
    mid = ((high + low) / 2).floor();

    if (arr[mid] == searchValue) {
      return mid;
    } else if (arr[mid] < searchValue) {
      low = mid + 1;
    } else if (arr[mid] > searchValue) {
      high = mid - 1;
    }
  }

  return -1;
}
```

[Link for Code]()

Part II:

Dynamic and Static Arrays

Types of Arrays:

There are basically two types of arrays:

- **Static Array:** In this type of array, memory is allocated at compile time having a fixed size of it. We cannot alter or update the size of this array.
- **Dynamic Array:** In this type of array, memory is allocated at run time but not having a fixed size. Suppose, a user wants to declare any random size of an array, then we will not use a static array, instead of that a dynamic array is used in hand. It is used to specify the size of it during the run time of any program.

Static Array

A =

44	12	-5	17	6	0	3	9	100
0	1	2	3	4	5	6	7	8

A[0] = 44

A[1] = 12

A[4] = 6

A[7] = 9

A[9] => index out of bounds!

Dynamic Array

Suppose we create a dynamic array with an initial capacity of two and then begin adding elements to it.

Ø	Ø

7	Ø

7	**-9**

7	-9	**3**	Ø

7	-9	3	**12**

7	-9	3	12	**5**	Ø	Ø	Ø

7	-9	3	12	5	**-6**	Ø	Ø

Complexity

	Static Array	Dynamic Array
Access	O(1)	O(1)
Search	O(n)	O(n)
Insertion	N/A	O(n)
Appending	N/A	O(1)
Deletion	N/A	O(n)

Static and Dynamic Array Implementation

```
int[8] array = {12, -3, 0, 59, 22, -107, 83, 4};
```

* it is not possible to represent static and dynamic arrays in Dart, thus all codes for these arrays are written in C++

```
DynamicArray(int initialCapacity)
{
    arr = new int[initialCapacity];
    capacity = initialCapacity;
    length = 0;
}
```

```
void append(int newItem)
{
    if (capacity == length)
    {
        grow();
    }

    arr[length] = newItem;

    length++;
}
```

```
void grow()
{
    int *newArr = new int[2 * capacity];
    for (int i = 0; i < capacity; i++)
    {
        newArr[i] = arr[i];
    }

    delete[] arr;
    arr = newArr;
    capacity *= 2;
}
```

```
void removeLast()
{
    arr[--length] = -1; // -1 is used to indicate empty slot in array as sentinel value
    if (capacity * 0.25 >= length)
    {
        shrink();
    }
}
```

* To avoid thrashing problem, which is caused when array is frequently resized, we will shrink array only when number of elements is less than or equal to 25% of array capacity

```
void shrink()
{
    int *newArr = new int[capacity / 2];
    for (int i = 0; i < length; i++)
    {
        newArr[i] = arr[i];
    }

    delete[] arr;
    arr = newArr;
    capacity /= 2;
}
```

[Link for full implementation]()

Thanks

References:

1) https://www.youtube.com/watch?v=XMUe3zFhM5c
2) https://www.codecademy.com/learn/cspath-asymptotic-notation/modules/cspath-asymptotic-notation/cheatsheet#
3) https://byjus.com/maths/asymptotes/
4) https://www.geeksforgeeks.org/how-do-dynamic-arrays-work/
5) https://www.youtube.com/watch?v=sP2AGTLROJs
6) https://www.geeksforgeeks.org/binary-search/