CSC 212
Data Structures & Algorithms

Fall 2022 | Jonathan Schrader

Dynamic Arrays

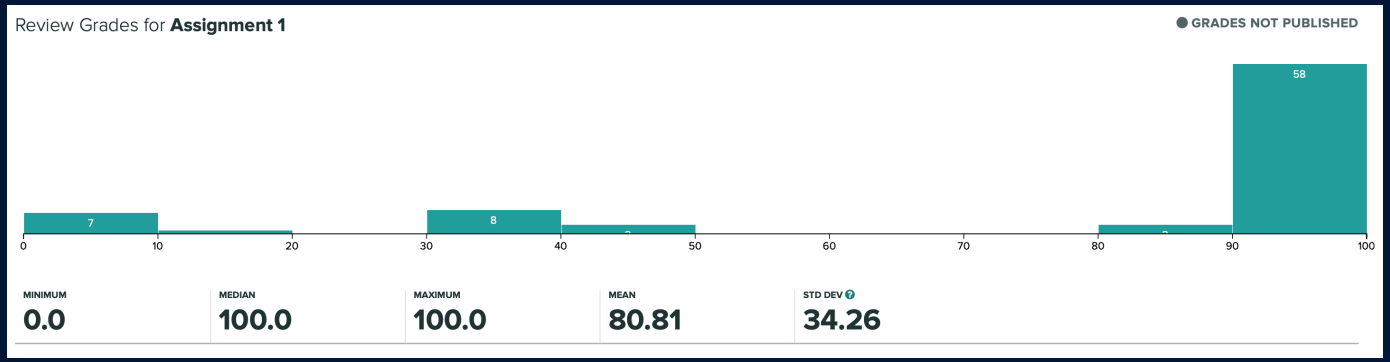
Housekeeping

This Week

Lab 3
- Dynamic Arrays

Assignment 2 [review/start…]
- Maze Development

Assignment 1 Outcomes



Input Size	Submissions	>= 70%	% Passing
90	80	60	75

“You miss 100% of the shots you don’t take…”

“look to your left, look to your right…”

True or False?

Time Complexities: $\{1, logn, n, nlogn, n^2, 2^n, n!\}$

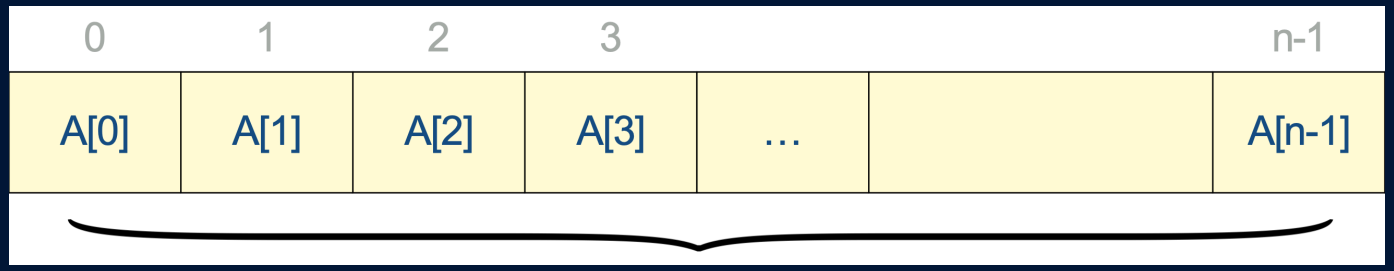
	Big O	Big Omega	Big Theta
$10^2 + 3000n + 10$	$\{n, nlogn, n^2, 2^n, n!\}$	$\{1, logn, n\}$	$\{n\}$
$21logn$	$\{logn, n, nlogn, n^2, 2^n, n!\}$	$\{1, logn\}$	$\{logn\}$
$500logn + n^4$	$\{n^2, 2^n, n!\}$	$\{1, logn, n, nlogn, n^2\}$	$\{n^2\}$
$\sqrt{n} + logn^{50}$	$\{n^2, 2^n, n!\}$	$\{1, logn, n, nlogn, n^2\}$	$\{n^2\}$
$4^n + n^{5000}$	$\{2^n, n!\}$	$\{1, logn, n, nlogn, n^2, 2^n\}$	$\{2^n\}$
$3000n^3 + n^{3.5}$	$\{n^2, 2^n, n!\}$	$\{1, logn, n, nlogn, n^2\}$	$\{n^2\}$
$2^5 + n!$	$n!$	$\{1, logn, n, nlogn, n^2, 2^n, n!\}$	$\{n!\}$

Arrays

An array is a contiguous sequence of elements of the *same type*

Each element can be accessed using *index*

- Array Name: A
- Array Length: n



note: all elements of the same data type

Declaration

```
//array declaration by specifying size
int myarray1[100];

//can also declare an array of user specified
//size (must be const for many compilers!)
int n = 8;
int myarray2[n];

// can declare and initialize elements
double arr[] = { 10.0, 20.0, 30.0, 40.0 };
// size is implicitly understood by the compiler when initialized at declaration

// alternative way
int arr[5] = { 1, 2, 3 };
// size is explicitly manipulated for the compiler
// size = 5, element count = 3, empty elements remaining = 2
```

Static arrays

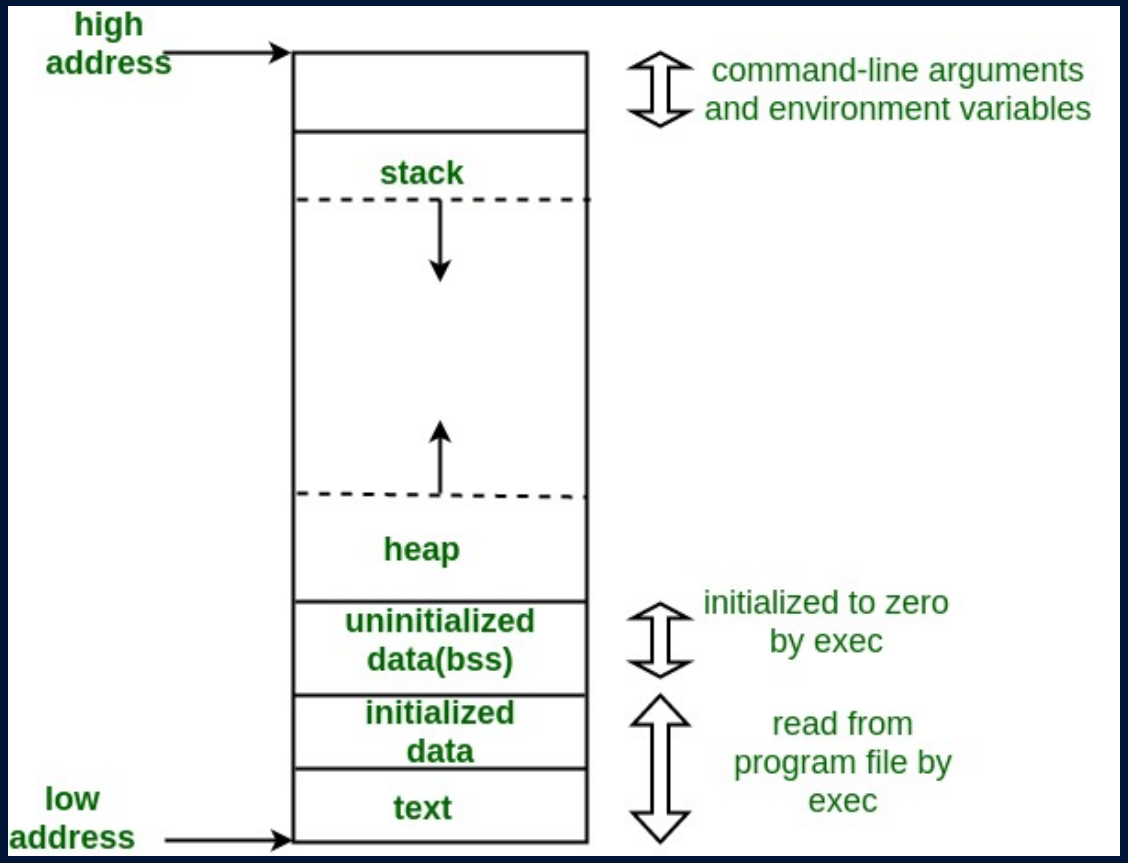
So far … we have seen examples of arrays, *allocated in the stack* (fixed length)

```
//array declaration by specifying size
int myarray1[100];
```

You can allocate memory dynamically, *allocated in the heap* (still fixed length)

```
int *myarray = new int[100];
//...
//work with the array
//...
delete []myarray;
```

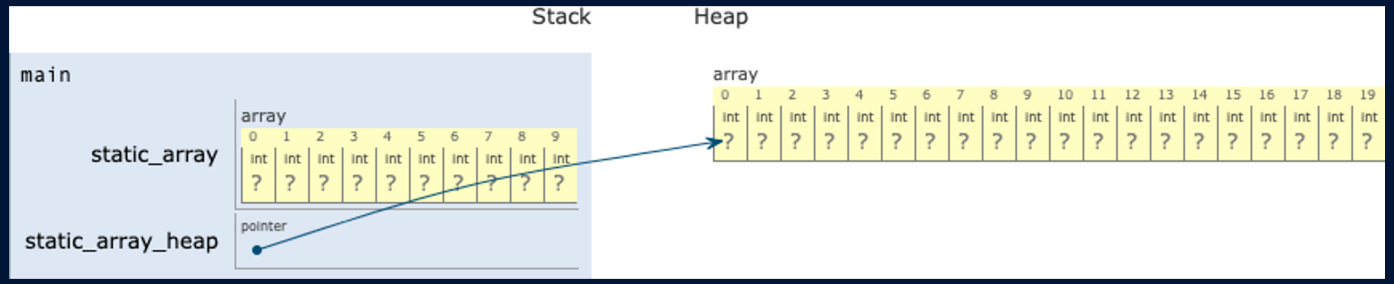
Memory layout of C/C++ programs



https://www.geeksforgeeks.org/memory-layout-of-c-program/

Memory Allocation

stack v. heap



Live Coding demo

Static Arrays

- stack and heap

Memory allocation, con't…

Parameter	STACK	HEAP
Basic	Memory is allocated in a contiguous block.	Memory is allocated in any random order.
Allocation and De-allocation	Automatic by compiler instructions.	Manual by the programmer.
Cost	Less	More
Implementation	Easy	Hard
Access time	Faster	Slower
Main Issue	Shortage of memory	Memory fragmentation
Locality of reference	Excellent	Adequate
Safety	Thread safe, data stored can only be accessed by owner	Not Thread safe, data stored visible to all threads
Flexibility	Fixed-size	Resizing is possible
Data type structure	Linear	Hierarchical
Preferred	Static memory allocation is preferred in array.	Heap memory allocation is preferred in the linked list.
Size	Small than heap memory.	Larger than stack memory.

What if…?

We don't know the max size of an array before running the program!

- user specified inputs/decisions!
- e.g. read an image or video and display

The sequence changes over time (during the execution of the program)

- e.g. you develop a text editor and represent the sequence of characters as anarray

Which data structure (studied so far) would you use on each case?

Dynamic Arrays (resizing, growing)

Dynamic Arrays

Dynamically allocated arrays that change their size over time

- can *grow* automatically
- can *shrink* automatically

Operations on arrays

- `append`
- `remove_last`
- `get` — $\Theta(1)$
- `set` — $\Theta(1)$

First try…

Start with an empty array

For every *append*:

- increase the size of the array by 1 then write the new element

For every *remove_last*:

- remove the last element and then decrease the size of the array by 1

Analyzing cost (grow by 1)

Count array accesses (reads and writes) of adding first n elements

- will ignore the cost of allocating/deallocating arrays

n	append	copy
1	1	1
2	2	2
...8	8	8
... 16	16	16
... 128	128	128

- each row indicates the number of *reads and writes* necessary for appending an element into an *existing array of length n*

$$n + \sum_{i=0}^{n-1} i^2 = n + n^2 - n$$

$$\Theta(n^2)$$

Analyzing cost (doubling array)

Count array accesses (reads and writes) of adding first $n = 2^i$ elements

- will ignore the cost of allocating/deallocating arrays

n	append	copy
1	1	1
2	1	2
...8	1	3
... 16	1	4
... 128	1	7

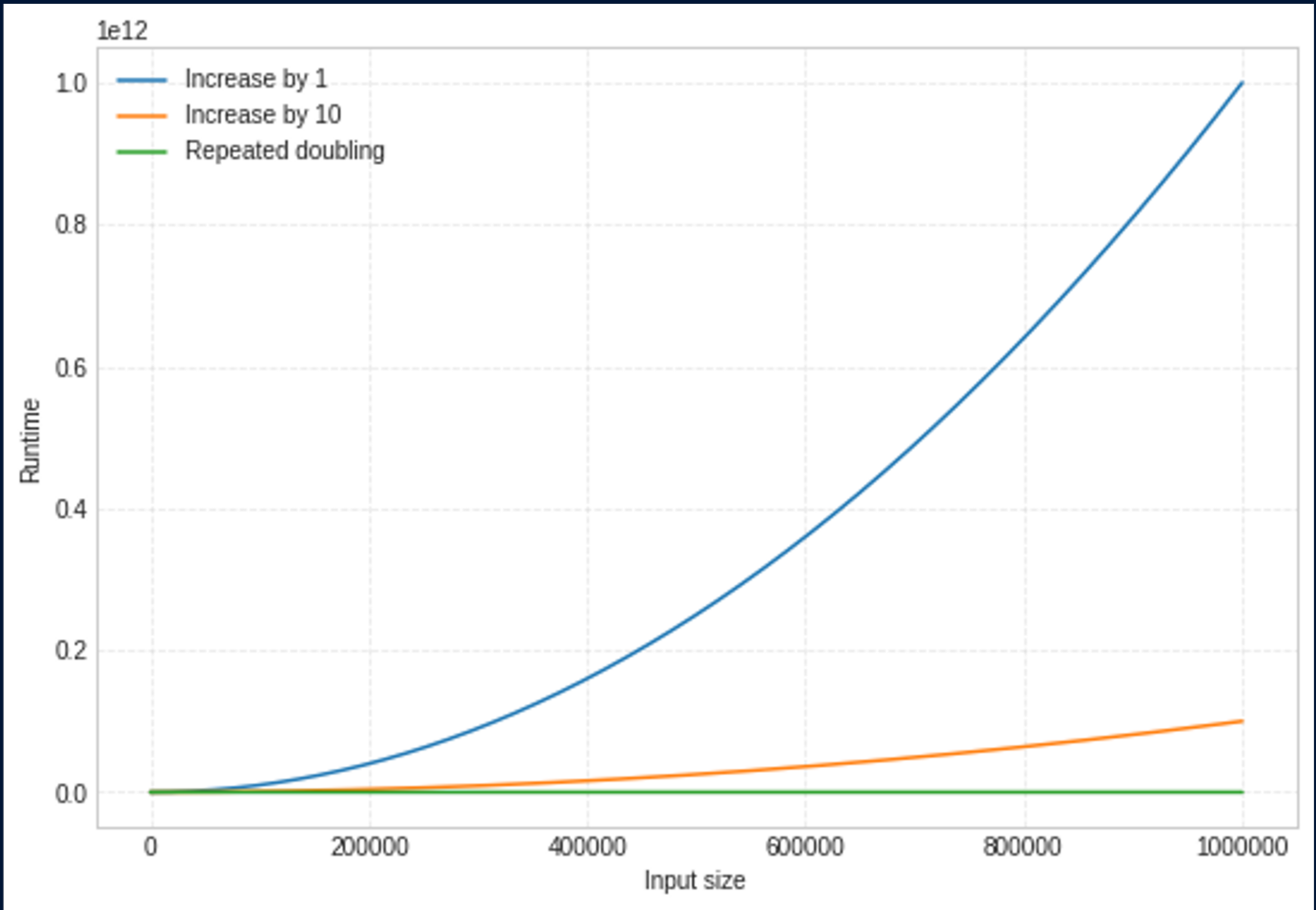
- each row indicates the number of *reads and writes* necessary for appending an element into an *existing array of length n*

$$n + \sum_{i=1}^{logn} 2^i = n + 2^{logn+1} - 1$$

$$\Theta(n)$$

$$\sum_{i=0}^{n} c^i = \frac{c^{n^2+1} - 1}{c - 1}$$

Proof


1e12
Increase by 1
Increase by 10
Repeated doubling
1.0
0.8
0.6
0.4
0.2
0.0
Runtime
0
200000
400000
600000
800000
1000000
Input size

Worst-case and average-case

Analysis for appending a *single element* using increase-by-1

Analysis for appending a *single element* using repeated doubling