COMP1819
Algorithms and Data Structures

Lecture 04: Searching – Linear and Binary

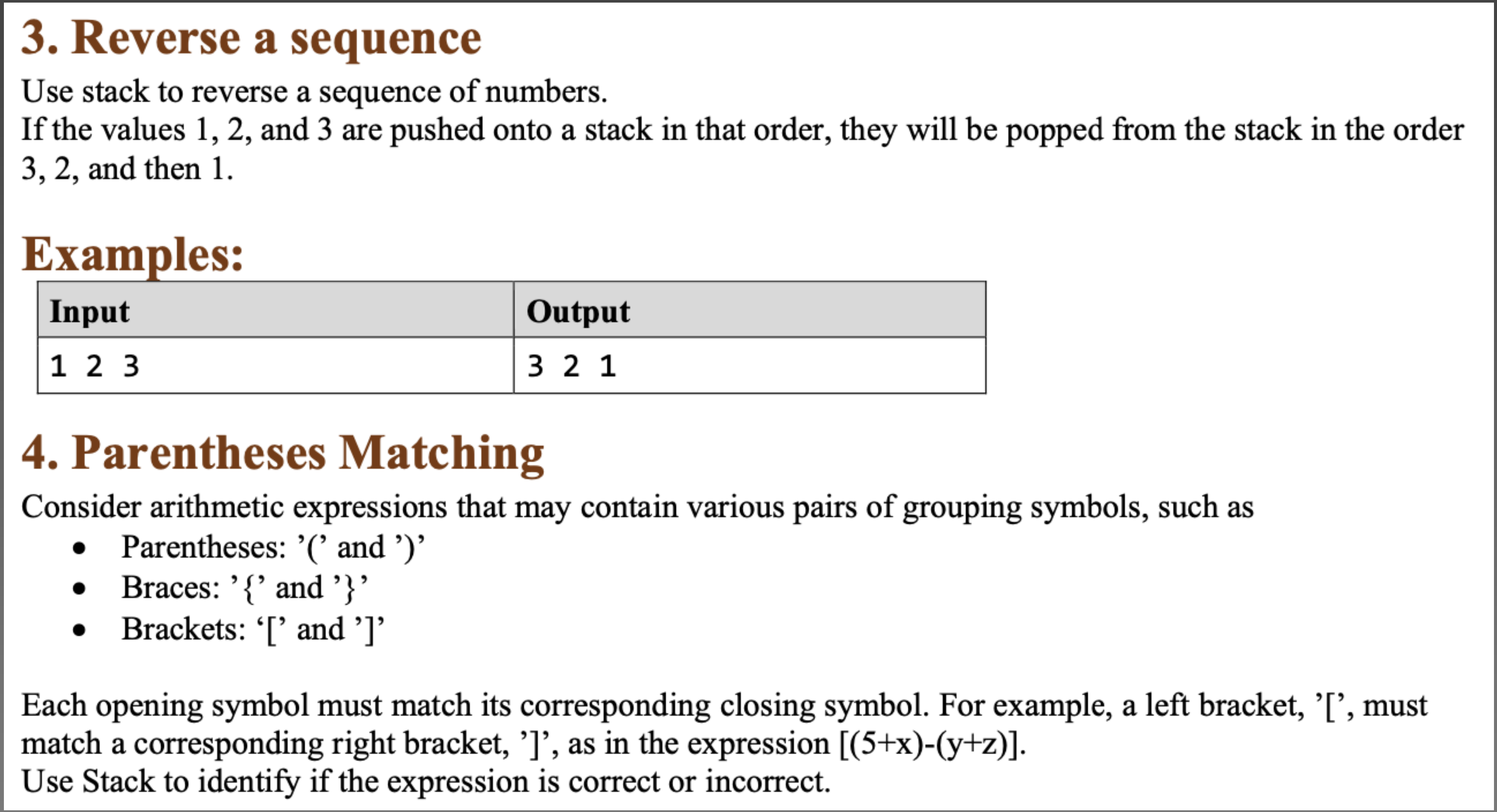
Dr. Tuan Vuong

09/02/2021

Content

- Lab 03
- Linear Search
- Binary Search
- Hashing
- Reinforcement

Lab 03



3. Reverse a sequence

Use stack to reverse a sequence of numbers.
If the values 1, 2, and 3 are pushed onto a stack in that order, they will be popped from the stack in the order 3, 2, and then 1.

Examples:

Input	Output
1 2 3	3 2 1

4. Parentheses Matching

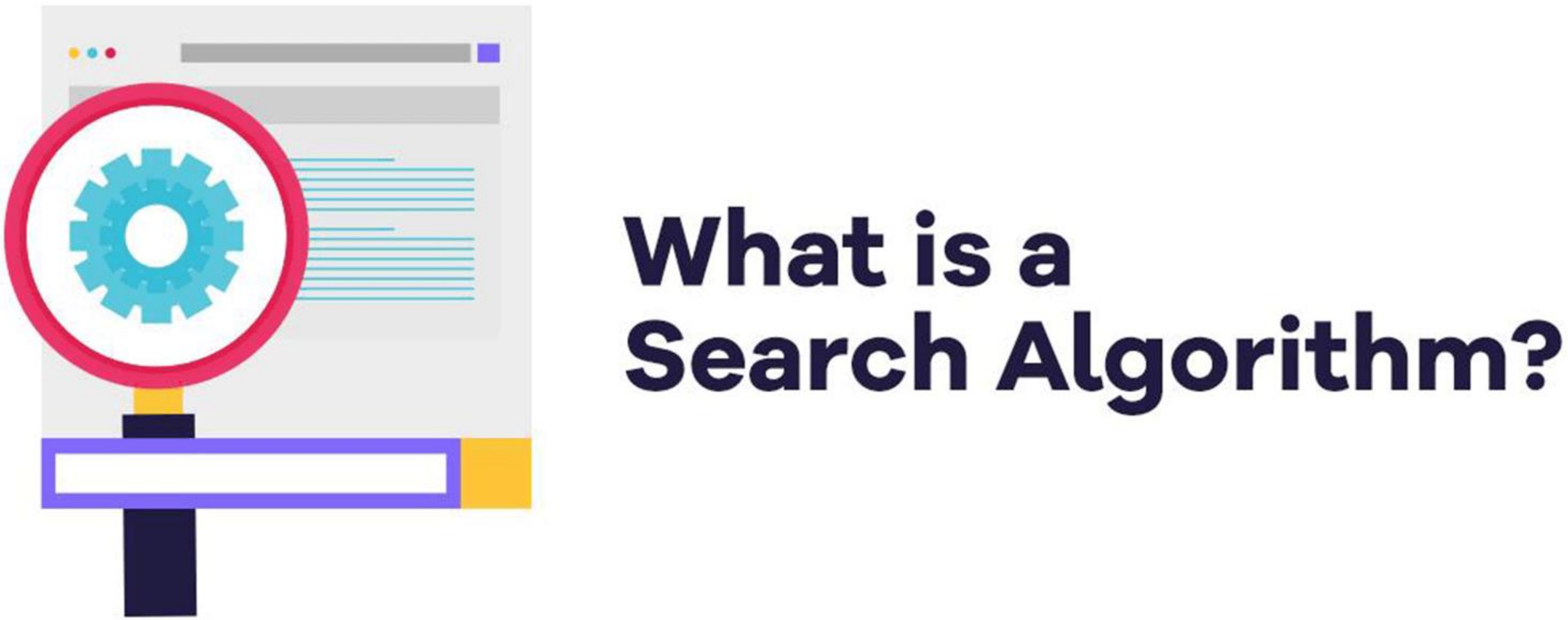
Consider arithmetic expressions that may contain various pairs of grouping symbols, such as

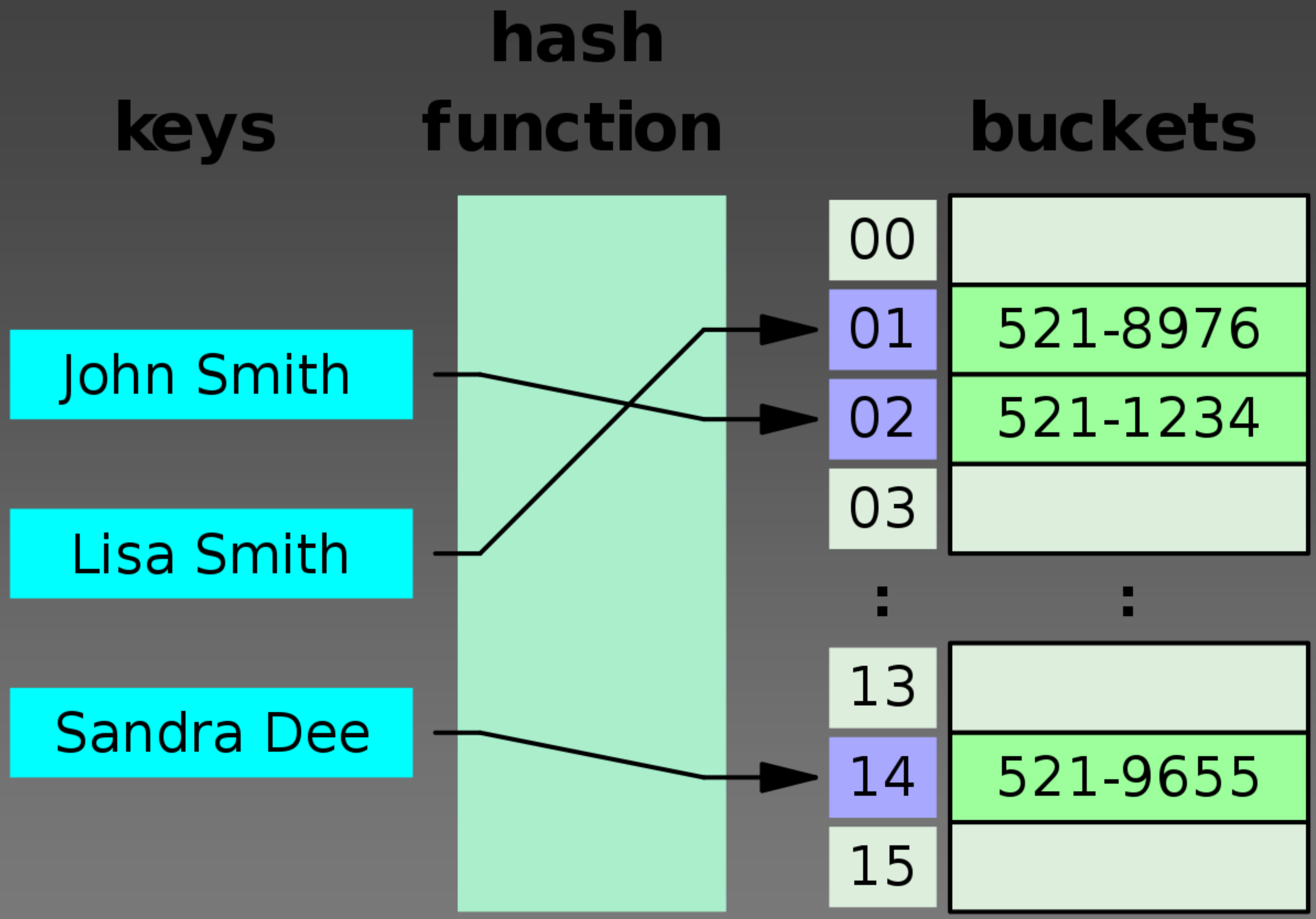
- Parentheses: ’(’ and ’)’
- Braces: ’{’ and ’}’
- Brackets: ‘[’ and ’]’

Each opening symbol must match its corresponding closing symbol. For example, a left bracket, ’[’, must match a corresponding right bracket, ’]’, as in the expression [(5+x)-(y+z)].
Use Stack to identify if the expression is correct or incorrect.

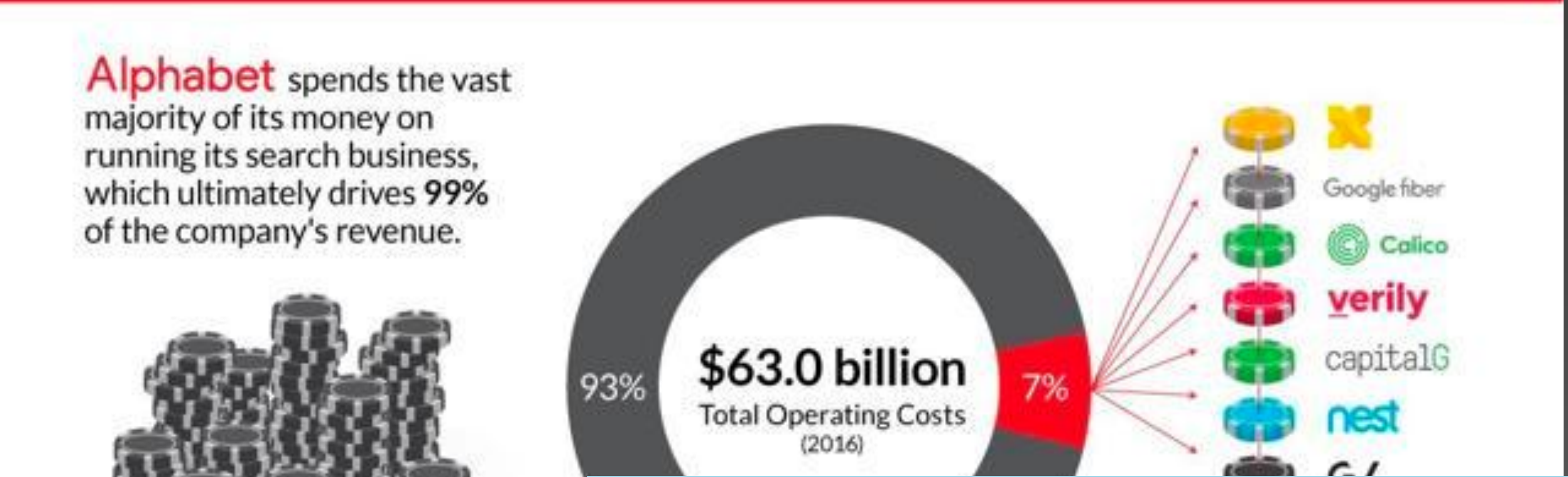
Solving problems with Stacks & Queues

Today

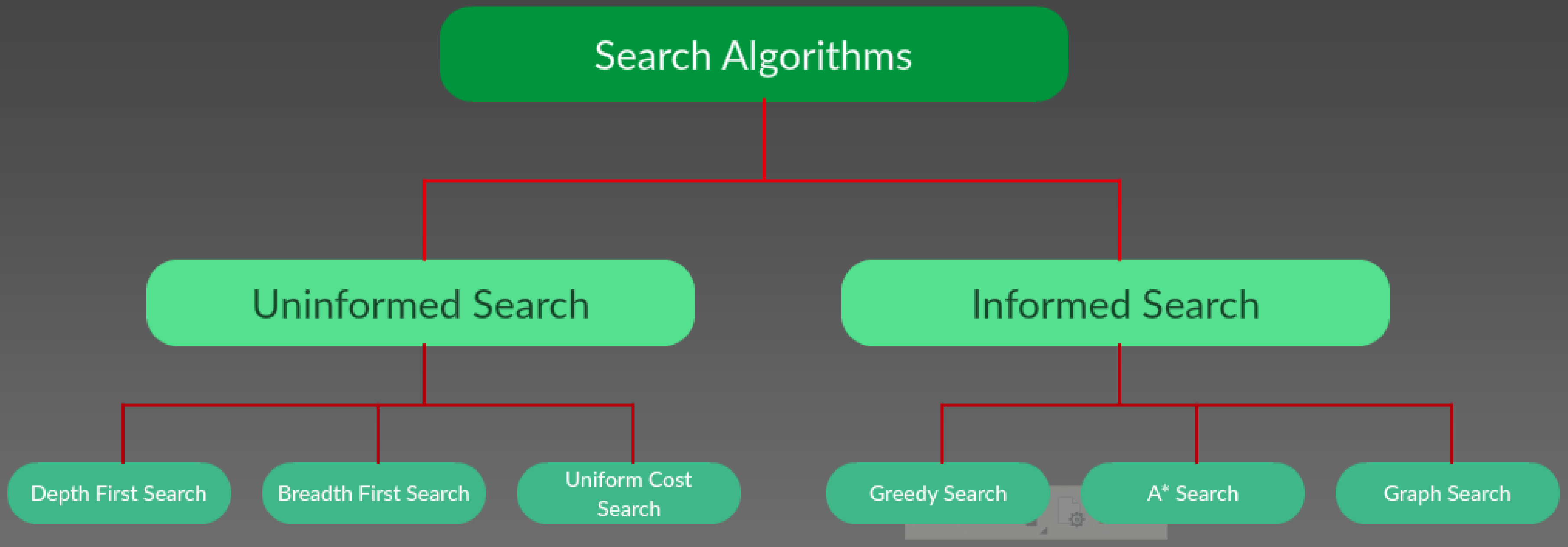




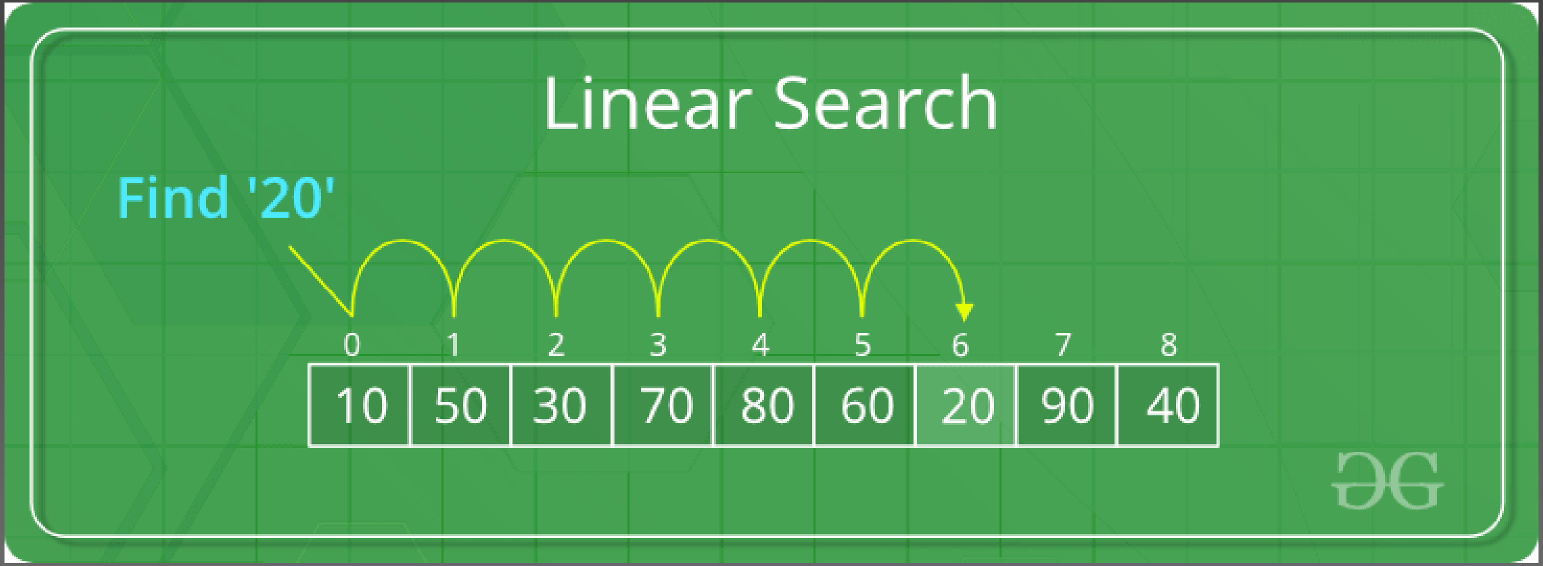
Search algorithm solves the search problem, namely, to retrieve information stored within some data structure, or calculated in the search space of a problem domain, either with discrete or continuous values



Why search? How does Google search works (https://www.youtube.com/watch?v=0eKVizvYSUQ)?

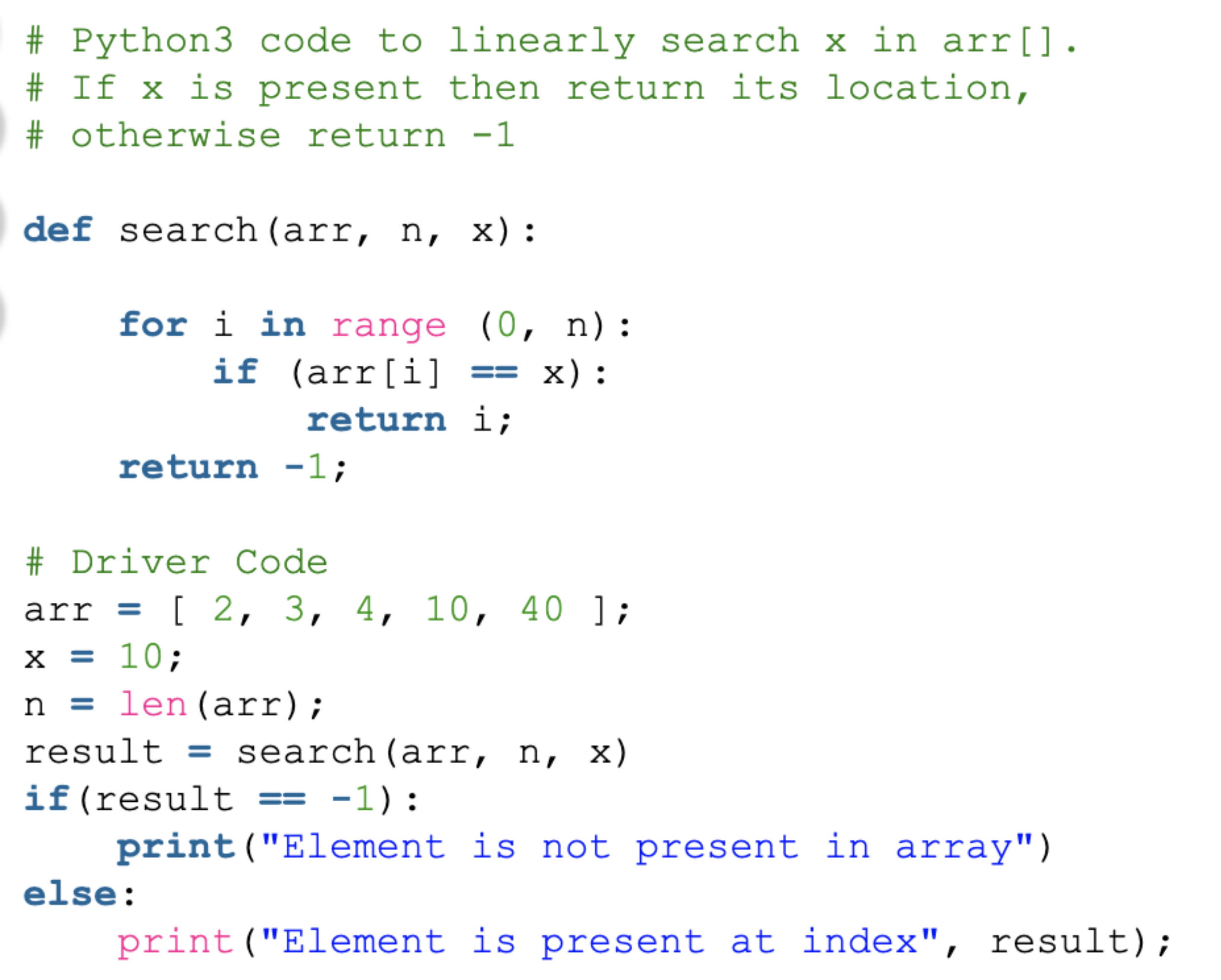


Search algorithms: Sequential Search, Binary Search



0	1	2	3	4	5	6	7	8
10	50	30	70	80	60	20	90	40

Sequential Search/Linear Search. How many comparisons would you need to do in order to find ‘20’?

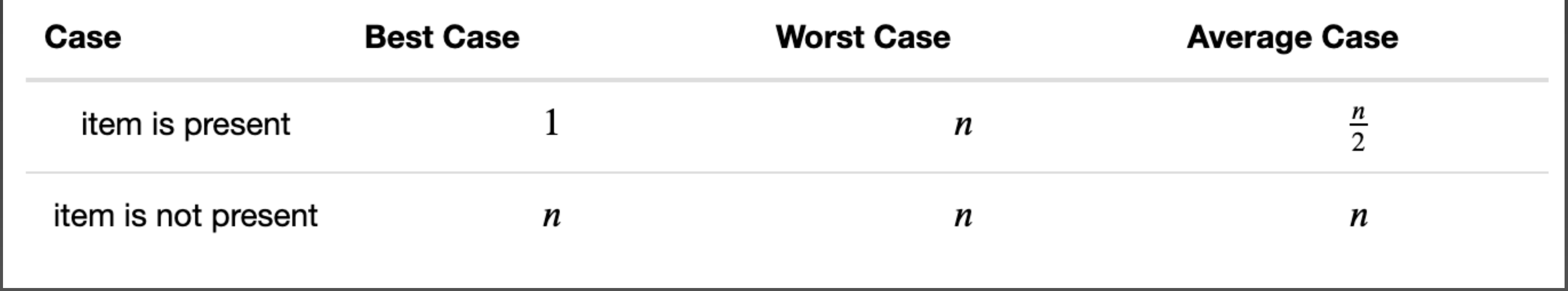


```
# Python3 code to linearly search x in arr[].
# If x is present then return its location,
# otherwise return -1

def search(arr, n, x):

    for i in range (0, n):
        if (arr[i] == x):
            return i;
    return -1;

# Driver Code
arr = [ 2, 3, 4, 10, 40 ];
x = 10;
n = len(arr);
result = search(arr, n, x)
if(result == -1):
    print("Element is not present in array")
else:
    print("Element is present at index", result);
```

Search algorithms: Sequential Search

Analysis of Algorithms

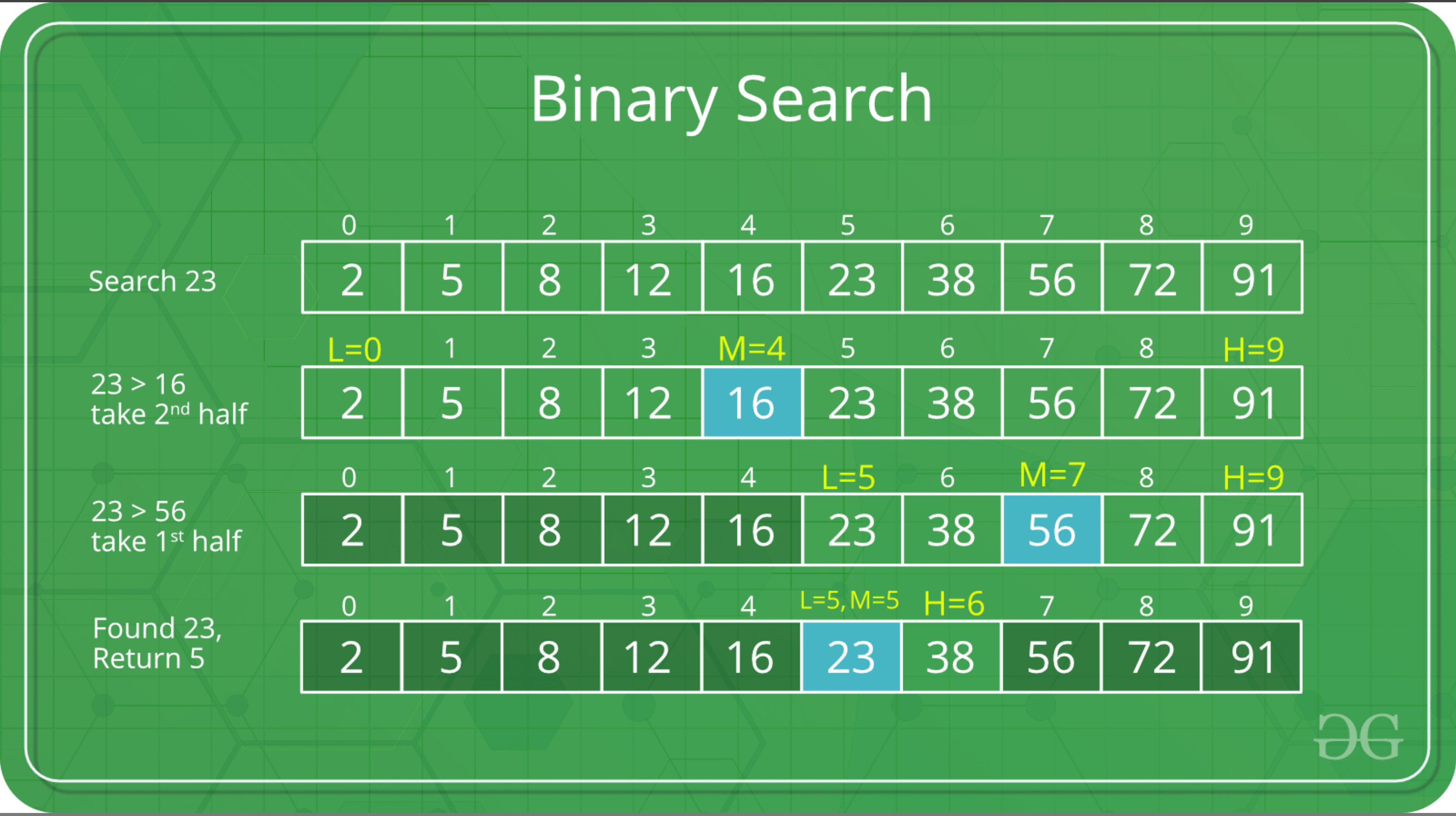
- In addition to describing the algorithms – analyse them
- What does analysis of algorithms involve ?
  - element comparisons
  - the number of element comparisons
- What is the best/worst/average case? When?



Case	Best Case	Worst Case	Average Case
item is present	1	n	$\frac{n}{2}$
item is not present	n	n	n

- Suppose you are doing a sequential search of the ordered list [3, 5, 6, 8, 11, 12, 14, 15, 17, 18]. How many comparisons would you need to do in order to find the key 11?

Is the list sorted? What if sorted?



Binary Search

Search 23

0	1	2	3	4	5	6	7	8	9
2	5	8	12	16	23	38	56	72	91

23 > 16
take 2nd half

L=0	1	2	3	M=4	5	6	7	8	H=9
2	5	8	12	16	23	38	56	72	91

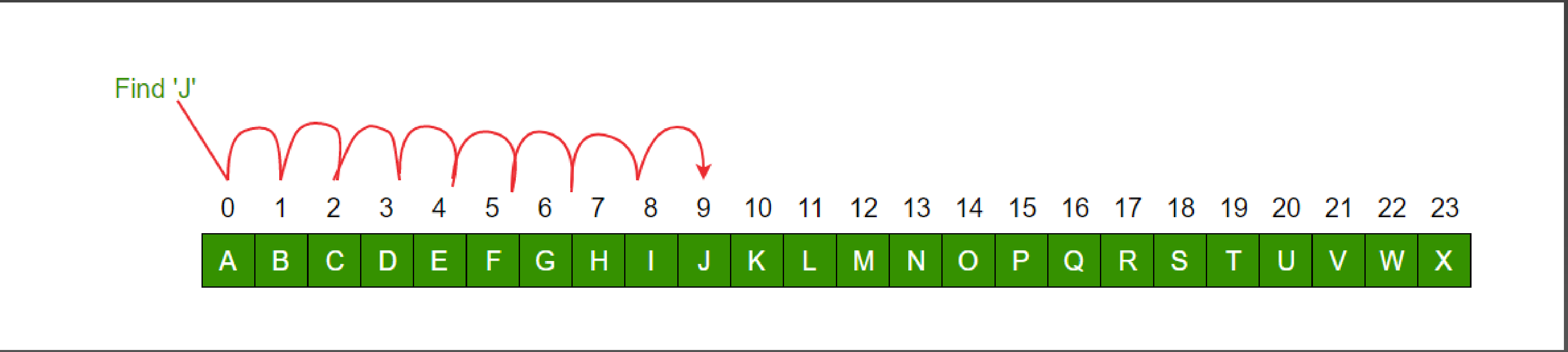
23 > 56
take 1st half

0	1	2	3	4	L=5	6	M=7	8	H=9
2	5	8	12	16	23	38	56	72	91

Found 23,
Return 5

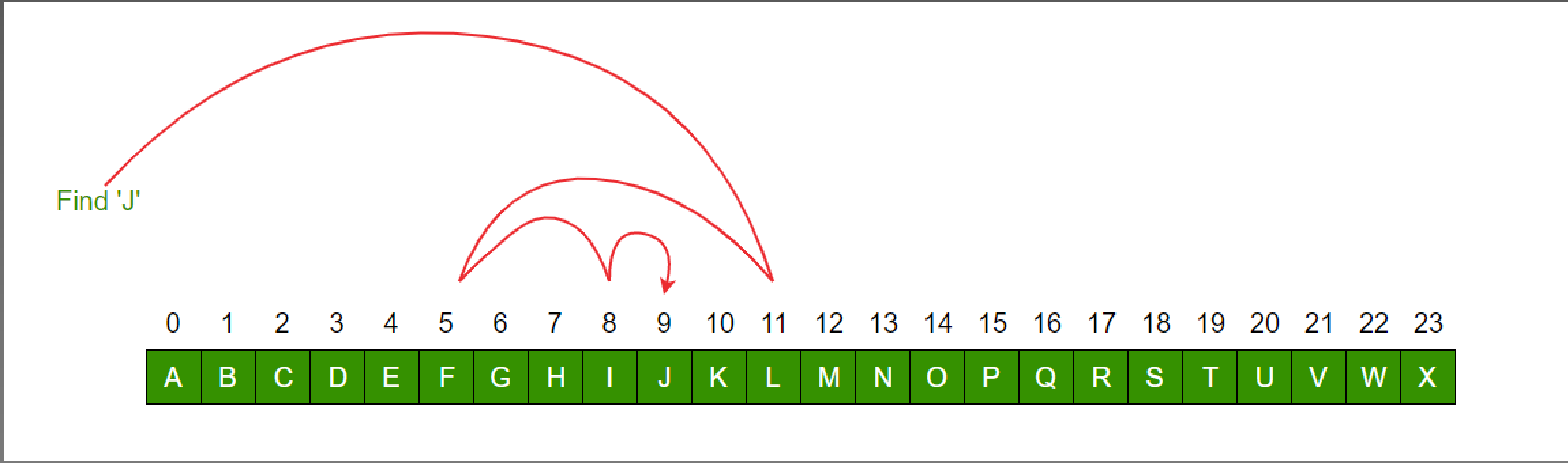
0	1	2	3	4	L=5, M=5	H=6	7	8	9
2	5	8	12	16	23	38	56	72	91

Binary Search on Sorted list.



Find 'J'

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X



Find 'J'

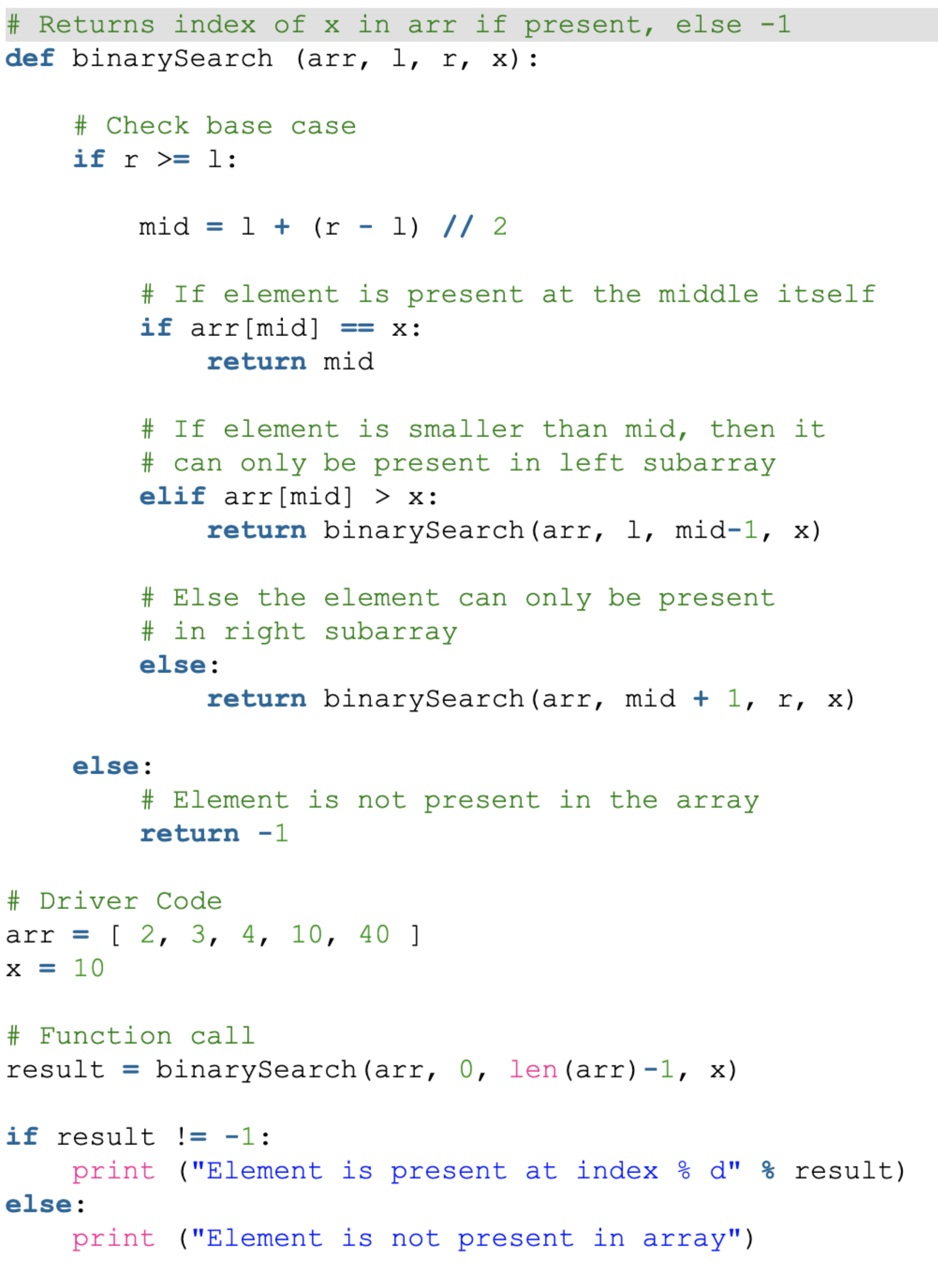
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X

Linear Search vs Binary Search

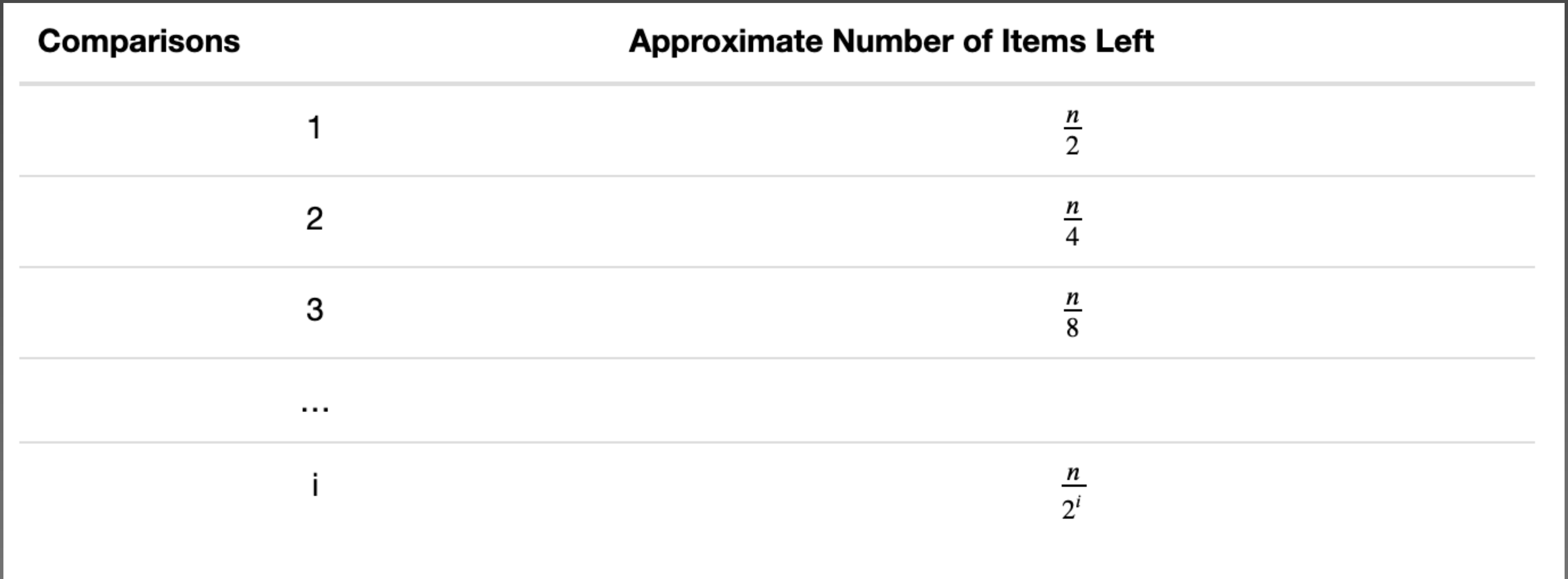
How about some recursive implementation?

Let’s debug.

```
Mid  = (right + left) // 2
Or same as left + (right –
left) // 2
```

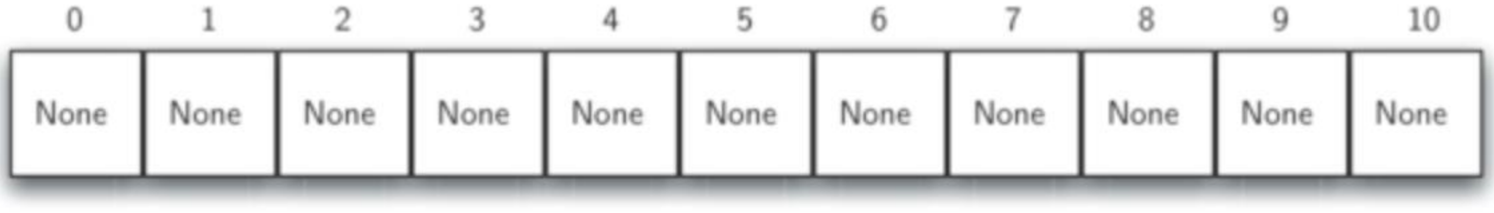


```python
# Returns index of x in arr if present, else -1
def binarySearch (arr, l, r, x):

    # Check base case
    if r >= l:

        mid = l + (r - l) // 2

        # If element is present at the middle itself
        if arr[mid] == x:
            return mid

        # If element is smaller than mid, then it
        # can only be present in left subarray
        elif arr[mid] > x:
            return binarySearch(arr, l, mid-1, x)

        # Else the element can only be present
        # in right subarray
        else:
            return binarySearch(arr, mid + 1, r, x)

    else:
        # Element is not present in the array
        return -1

# Driver Code
arr = [ 2, 3, 4, 10, 40 ]
x = 10

# Function call
result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:
    print ("Element is present at index % d" % result)
else:
    print ("Element is not present in array")
```



Comparisons	Approximate Number of Items Left
1	$\frac{n}{2}$
2	$\frac{n}{4}$
3	$\frac{n}{8}$
...	
i	$\frac{n}{2^i}$

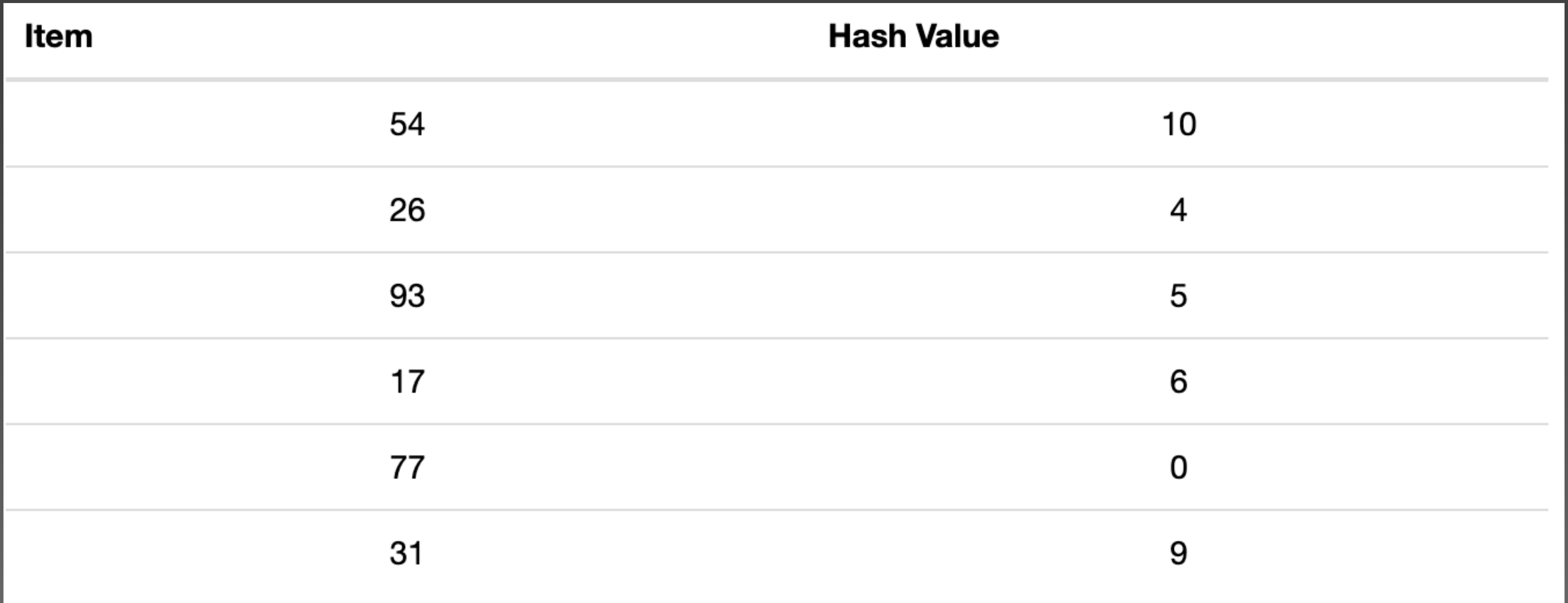
Algorithm Complexity ? O (log n)



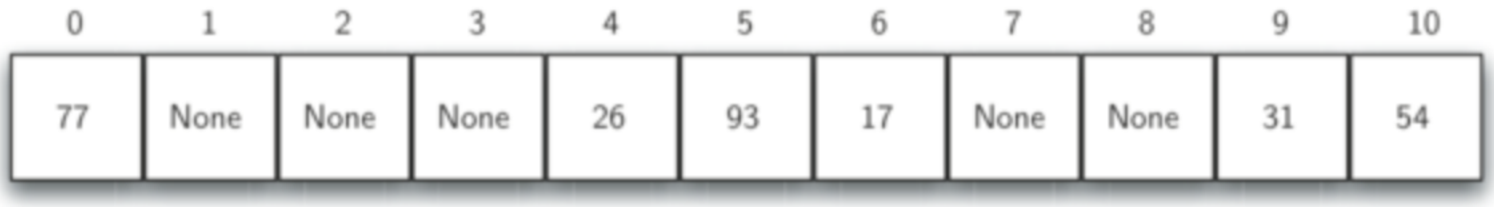
0	1	2	3	4	5	6	7	8	9	10
None	None	None	None	None	None	None	None	None	None	None

Figure 4: Hash Table with 11 Empty Slots

Improve further? Have you heard of “Hashing”?
Insert the following numbers into the Hash table: 54, 26, 93, 17, 77, and 31.



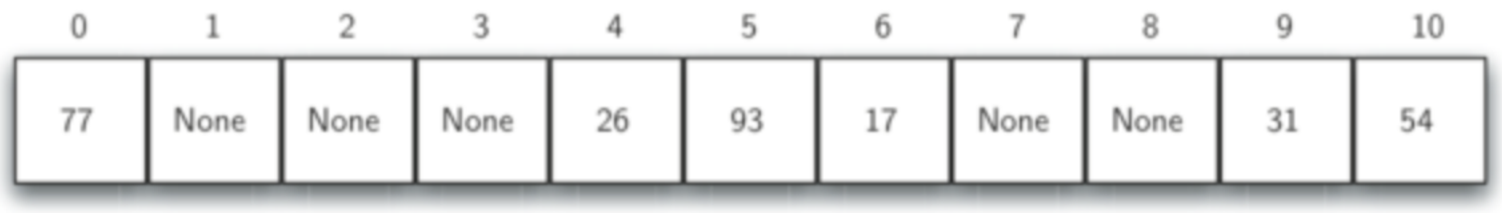
Item	Hash Value
54	10
26	4
93	5
17	6
77	0
31	9



0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

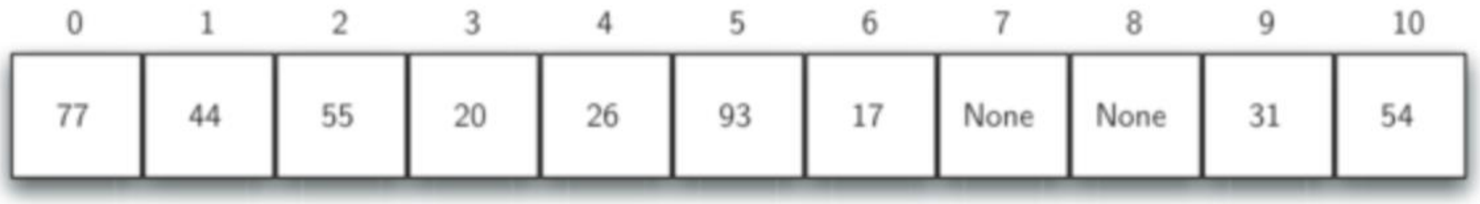
Figure 5: Hash Table with Six Items

Insert the following numbers into the Hash table: 54, 26, 93, 17, 77, and 31, using “remainder method” ~ $h(item)=item\%11$



0	1	2	3	4	5	6	7	8	9	10
77	None	None	None	26	93	17	None	None	31	54

Figure 5: Hash Table with Six Items

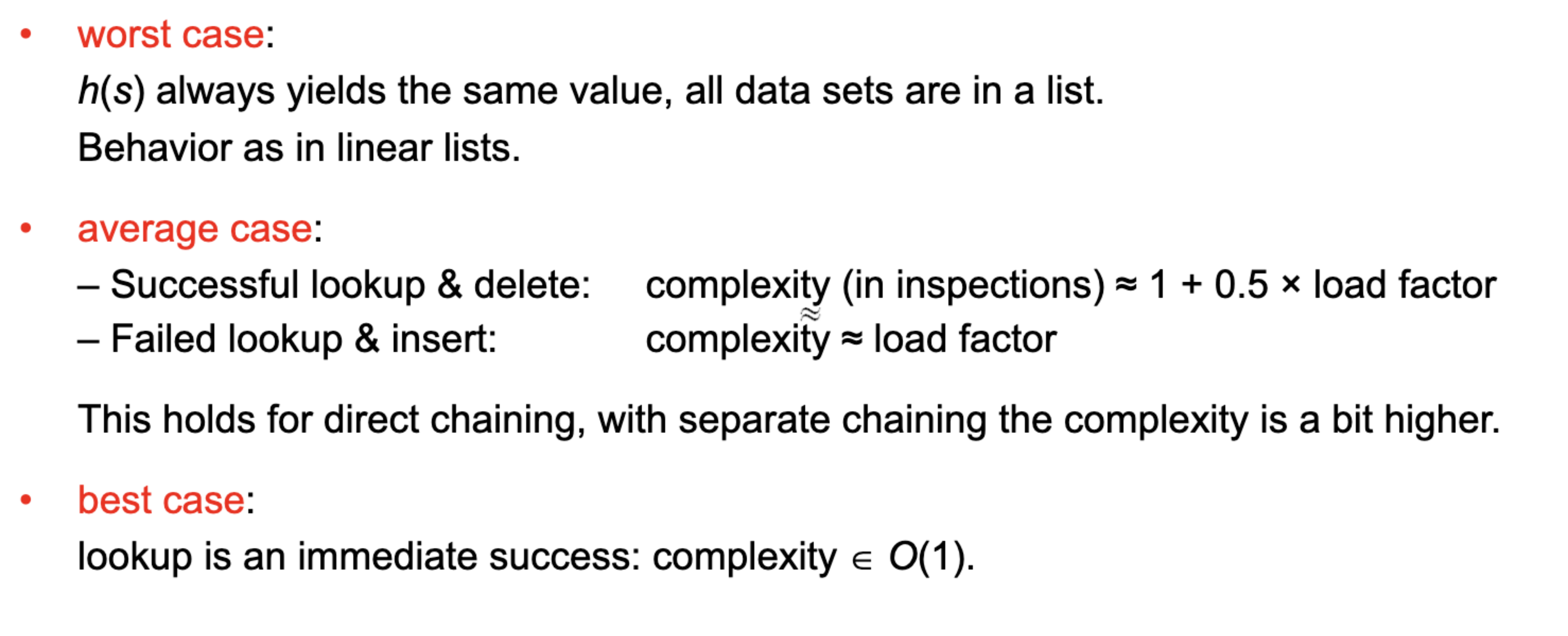


0	1	2	3	4	5	6	7	8	9	10
77	44	55	20	26	93	17	None	None	31	54

Figure 8: Collision Resolution with Linear Probing

Collision when inserting 44, 45, 20 into the Hash table.
Collision Resolution with Linear Probing

Analysis of Hashing



- worst case:

  $h(s)$ always yields the same value, all data sets are in a list.

  Behavior as in linear lists.

- average case:

  – Successful lookup & delete: complexity (in inspections) ≈ 1 + 0.5 × load factor

  – Failed lookup & insert: complexity ≈ load factor

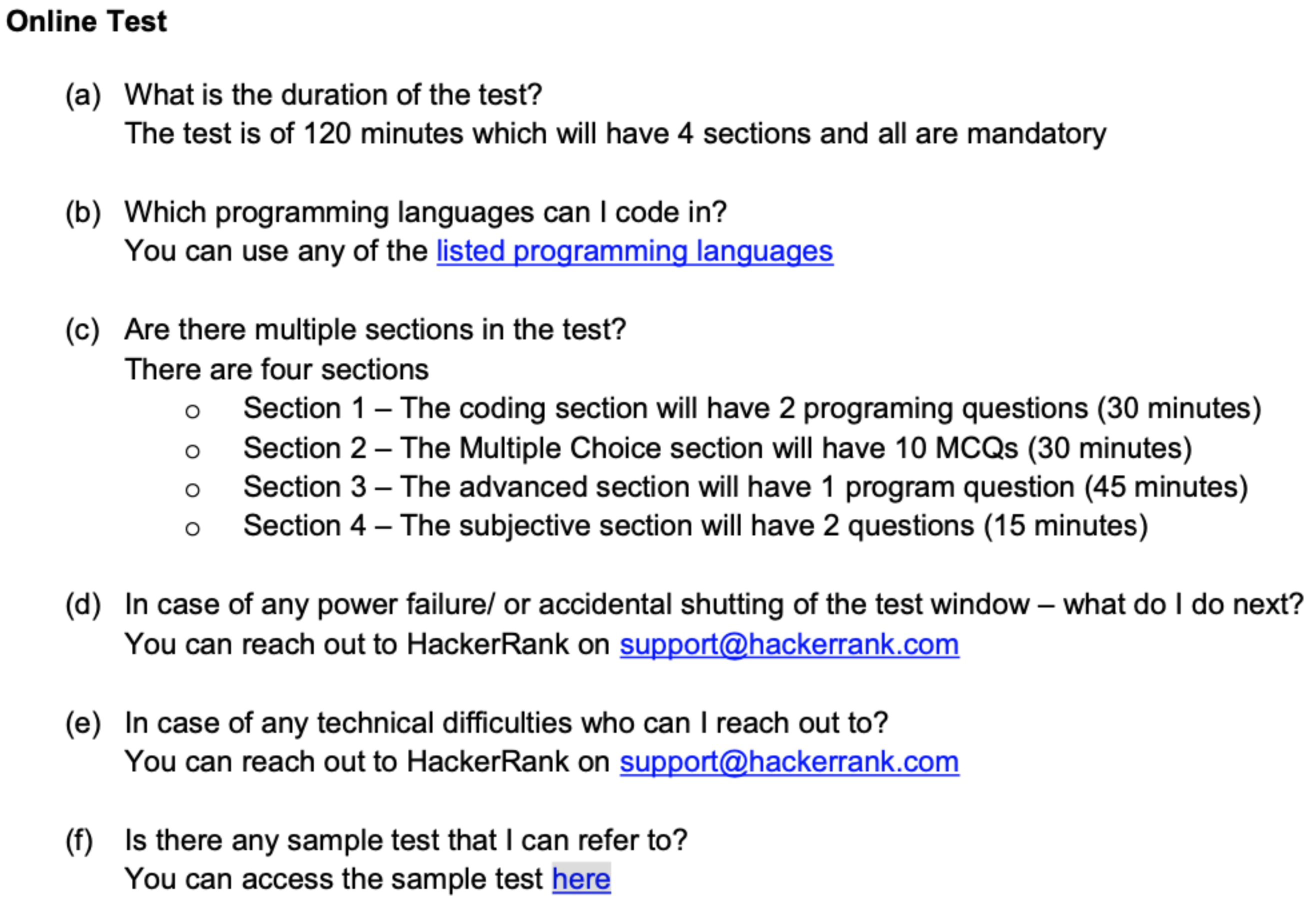
  This holds for direct chaining, with separate chaining the complexity is a bit higher.

- best case:

  lookup is an immediate success: complexity $\in O(1)$.

Reinforcement

Recruiting



Online Test

(a) What is the duration of the test?
The test is of 120 minutes which will have 4 sections and all are mandatory

(b) Which programming languages can I code in?
You can use any of the listed programming languages

(c) Are there multiple sections in the test?
There are four sections

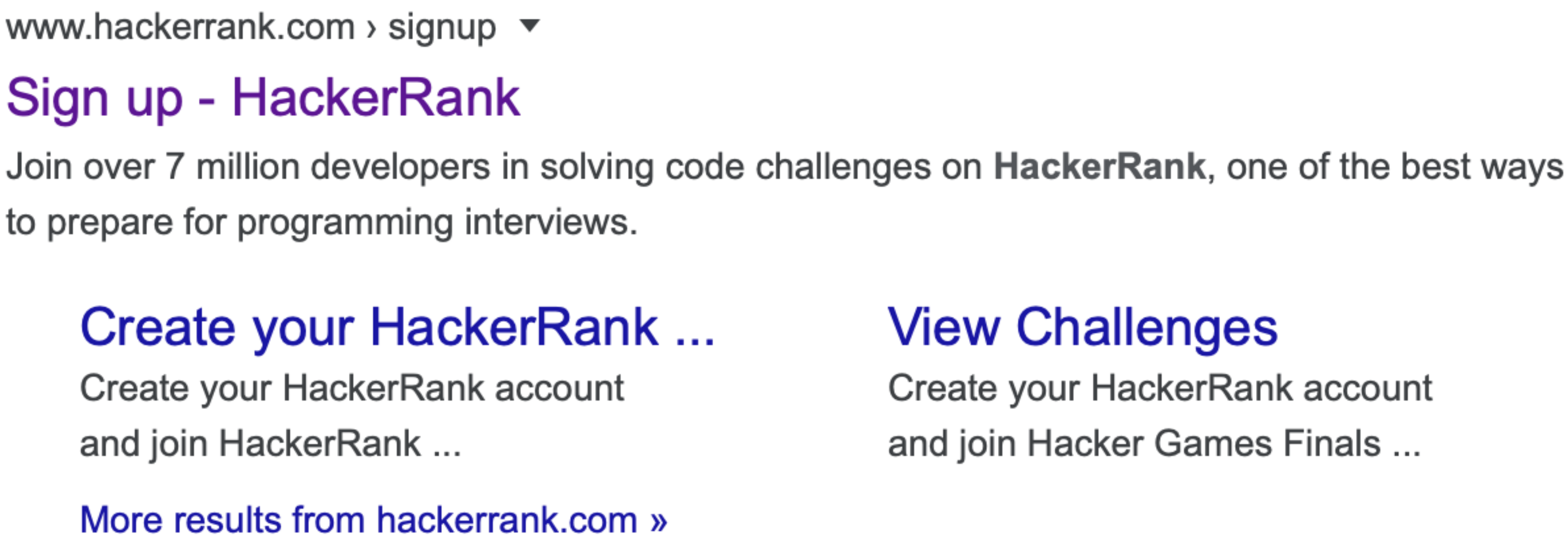
- Section 1 – The coding section will have 2 programing questions (30 minutes)
- Section 2 – The Multiple Choice section will have 10 MCQs (30 minutes)
- Section 3 – The advanced section will have 1 program question (45 minutes)
- Section 4 – The subjective section will have 2 questions (15 minutes)

(d) In case of any power failure/ or accidental shutting of the test window – what do I do next?
You can reach out to HackerRank on support@hackerrank.com

(e) In case of any technical difficulties who can I reach out to?
You can reach out to HackerRank on support@hackerrank.com

(f) Is there any sample test that I can refer to?
You can access the sample test here

Signup



www.hackerrank.com › signup ▾

Sign up - HackerRank

Join over 7 million developers in solving code challenges on **HackerRank**, one of the best ways to prepare for programming interviews.

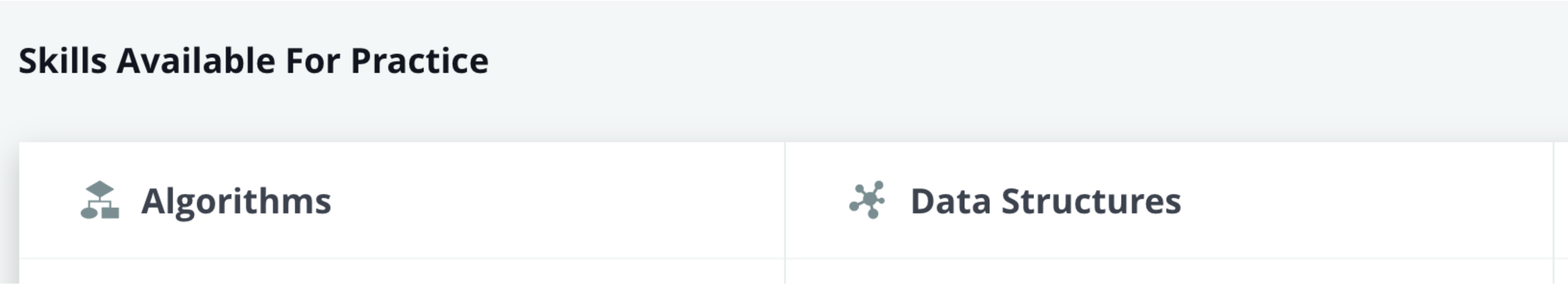
Create your HackerRank ...

Create your HackerRank account and join HackerRank ...

View Challenges

Create your HackerRank account and join Hacker Games Finals ...

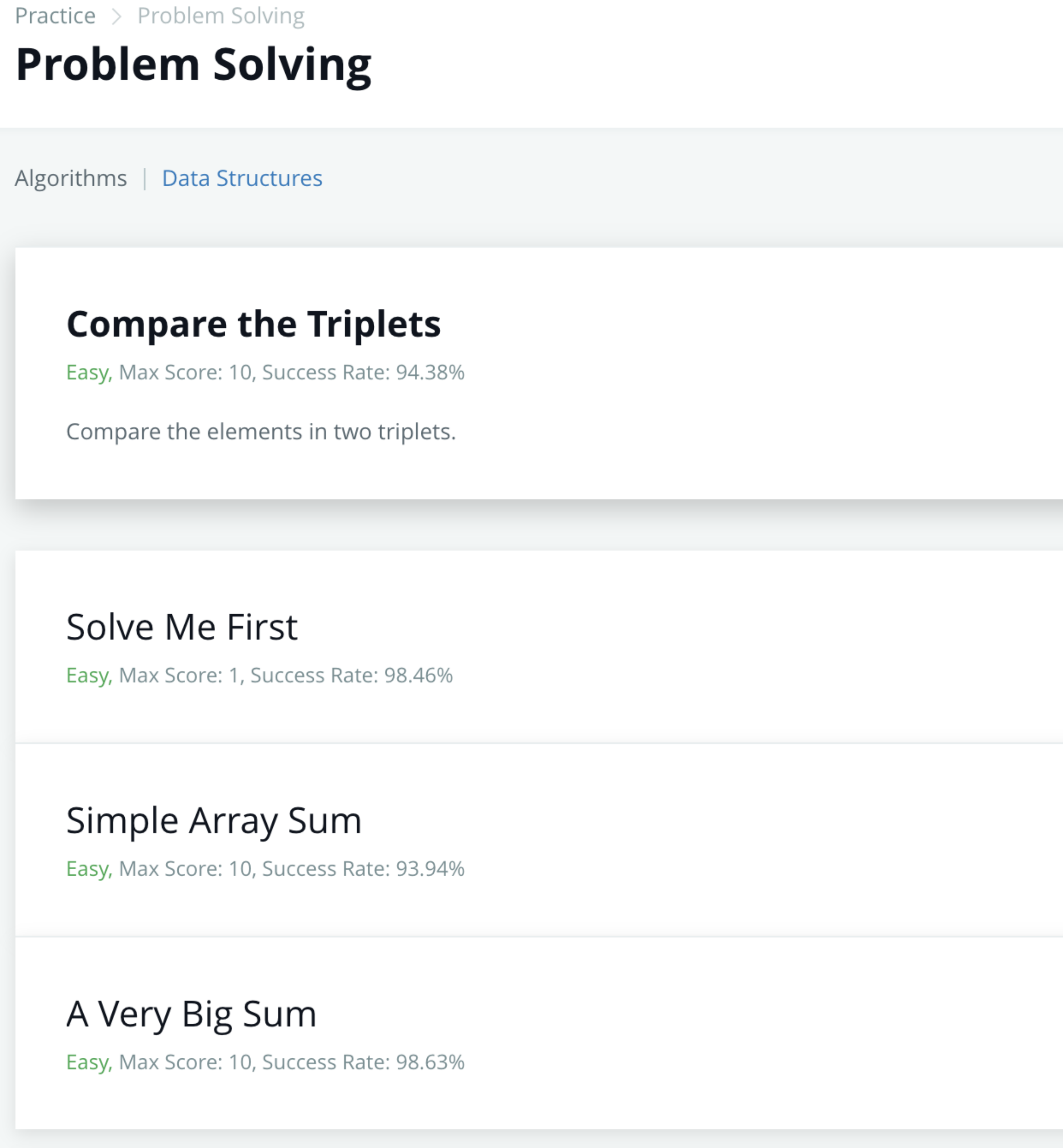
More results from hackerrank.com »



Skills Available For Practice

Algorithms

Data Structures



Practice > Problem Solving

Problem Solving

Algorithms | Data Structures

Compare the Triplets

Easy, Max Score: 10, Success Rate: 94.38%

Compare the elements in two triplets.

Solve Me First

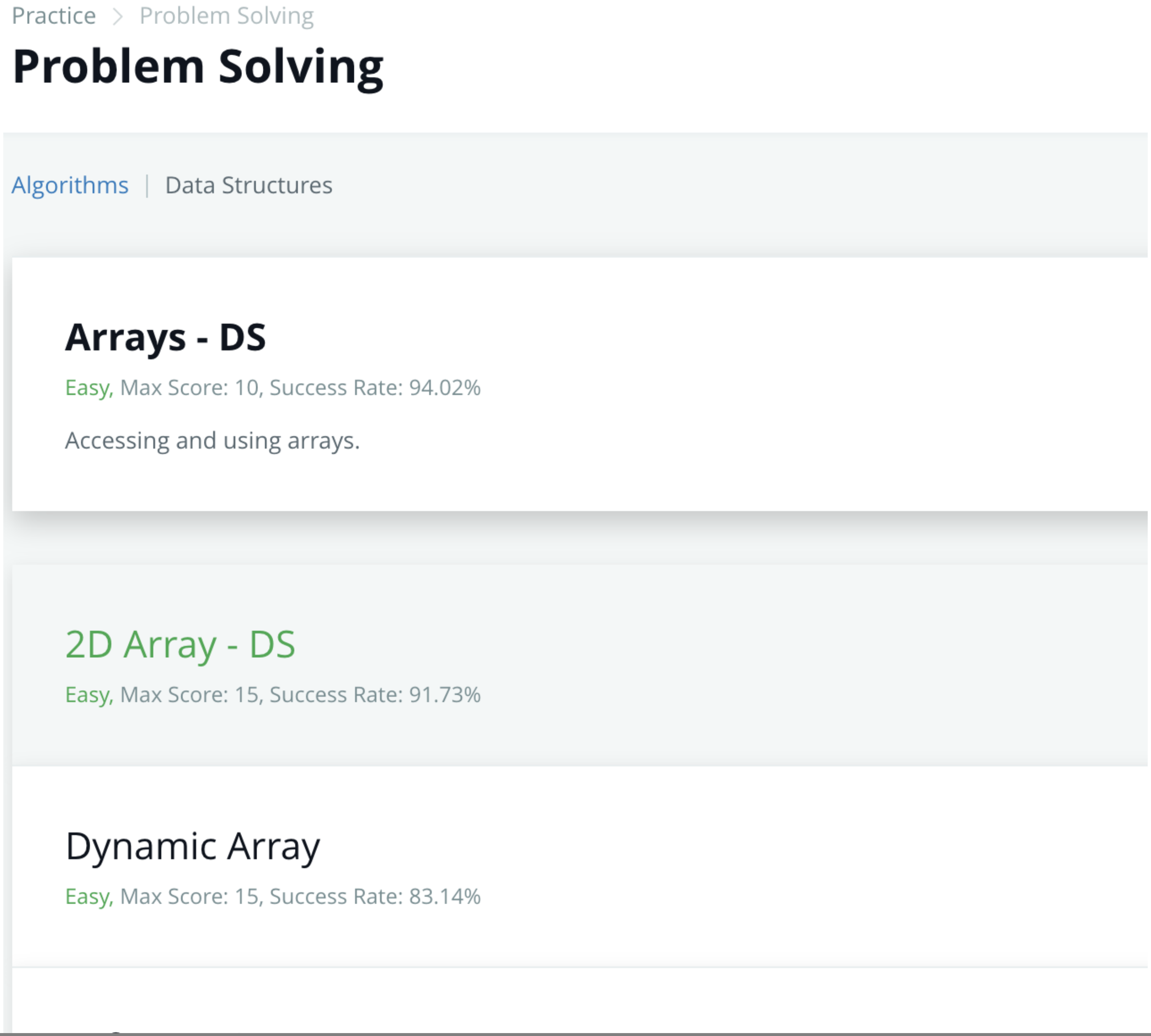
Easy, Max Score: 1, Success Rate: 98.46%

Simple Array Sum

Easy, Max Score: 10, Success Rate: 93.94%

A Very Big Sum

Easy, Max Score: 10, Success Rate: 98.63%



Practice > Problem Solving

Problem Solving

Algorithms | Data Structures

Arrays - DS

Easy, Max Score: 10, Success Rate: 94.02%

Accessing and using arrays.

2D Array - DS

Easy, Max Score: 15, Success Rate: 91.73%

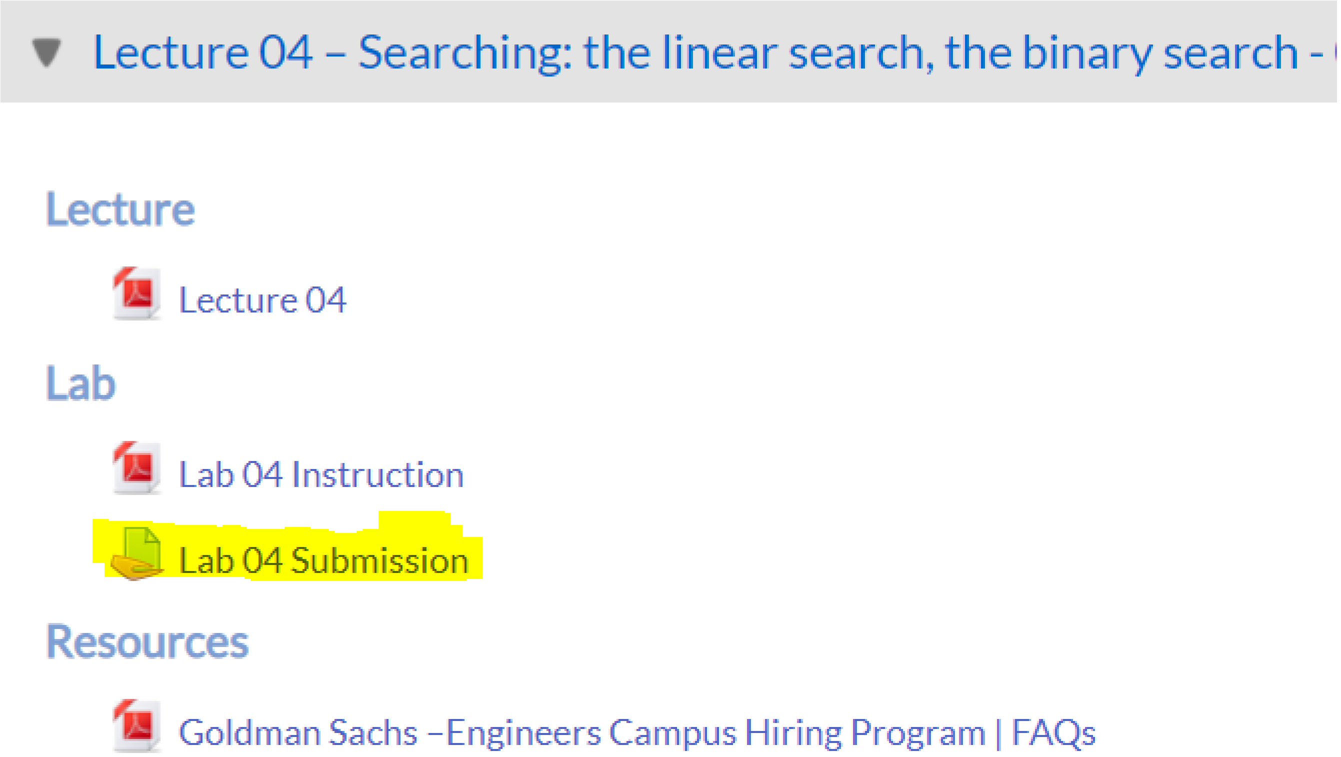
Dynamic Array

Easy, Max Score: 15, Success Rate: 83.14%

Left Rotation

Easy, Max Score: 20, Success Rate: 87.68%

These will help to complete the coursework.



▼ Lecture 04 – Searching: the linear search, the binary search -

Lecture

Lecture 04

Lab

Lab 04 Instruction

Lab 04 Submission

Resources

Goldman Sachs –Engineers Campus Hiring Program | FAQs

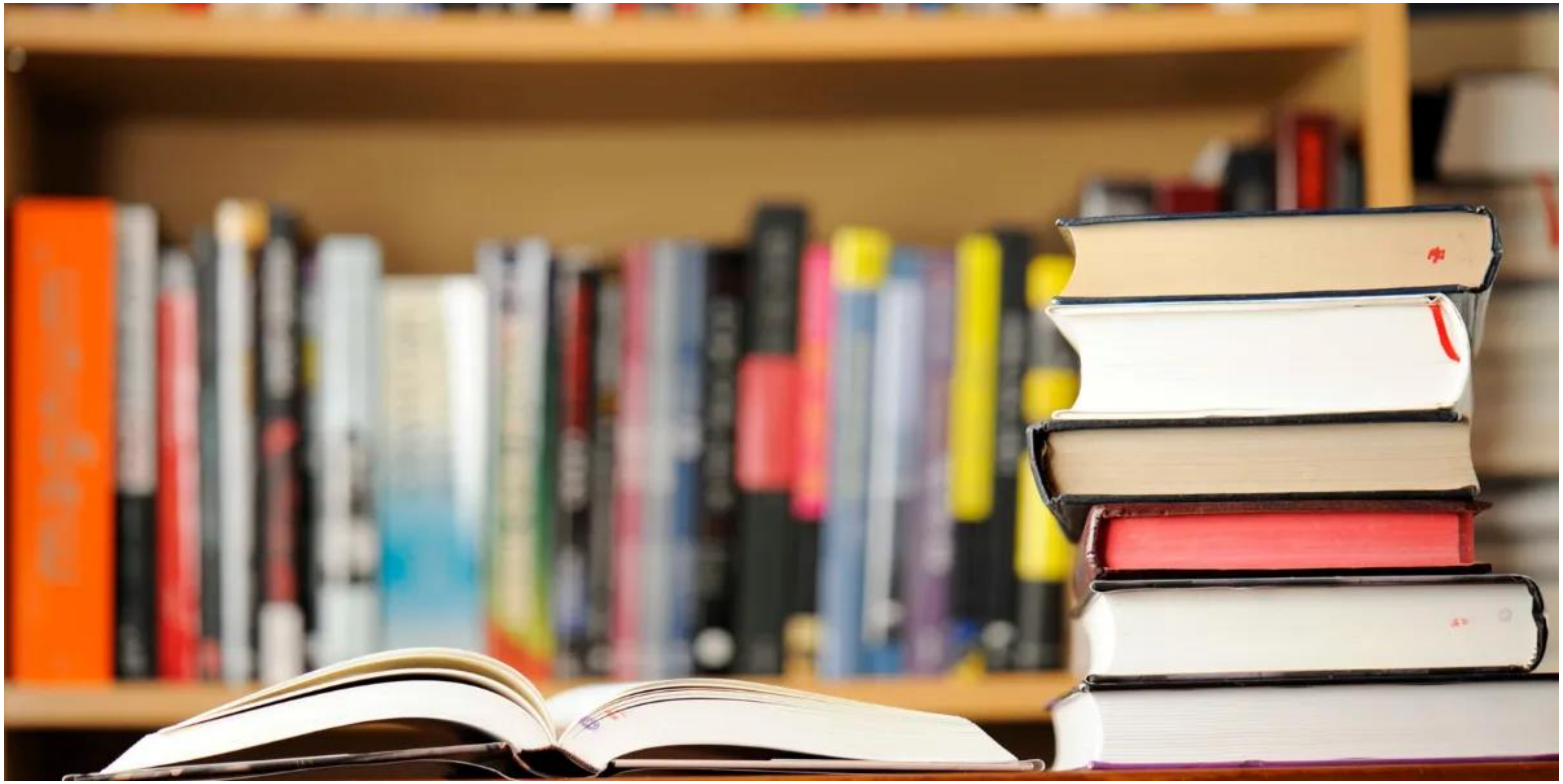
Please submit the screenshots by 16/02/2021. Instruction in Lab 04.

Quick overview

- A sequential search is O(n) for ordered and unordered lists.
- A binary search of an ordered list is O(log n) in the worst case.
- Hash tables can provide constant time searching.
- Join Hackerrank

Extra reading

- Implementation of Hashing with Python



Next week