Data Structures and Algorithms

Comp 200

Fall 2022



Department of Computer Science

Forman Christian College University

Lab 3

Big O Notation and Time Complexity

|  |  |
| --- | --- |
| **DESCRIPTION** | **MARKS ALLOCATED** |
| Attendance | 25% |
| Task Completion | 35% |
| Viva | 35% |
| Submission | 15% |

### Marks will be deducted in case if students have not completed the assigned task.

**Note that these marks are max in each category. We may assign less than the given percentage of marks in case students have not successfully completed all the requirements.**

**This lab is time constrained. Please note that you must finish and submit your work within given time**.

**Question 1**:

Find the time complexity of following function.

def useless\_function(array): 1

total = 0 1

return total 1

= 3

O(n)

**Question 2**:

Find the time complexity of following function.

def natural\_numbers(n): 1

for i in range(n): n

print(i+1) n

= 2n+1

O(n)

**Question 3**:

Find the time complexity of following code.

def find\_sum(given\_array): 1

total = 0 1

for i in given\_array: n

total += I 2n

return total 1

3n + 3

O(n)

**Question 4**:

Find the time complexity of following code.

def compute\_hcf(x, y): 1

if x > y: 1

smaller = y 1

else:

smaller = x

for i in range(1, smaller+1): 2n

if((x % i == 0) and (y % i == 0)): 5n

hcf = i n

return hcf 1

8n +4

O(n)

**Question 5**:

Find the time complexity of following code.

def pyramid(rows): 1

for i in range(rows): n

for j in range(i+1): 2n^2

print("\* ", end="") n^2

print("\n") n

2n + 3n^2 + 1

O(n^2)

**Question 6**:

Find the time complexity of following code.

def find\_sum\_2d\_array(array\_2d): 1

total = 0 1

for row in array\_2d: n

for i in row: n^2

total += I 2n^2

return total 1

n + 3n^2 + 3

O(n^2)

**Question 7**:

Find the time complexity of following code.

count = 0 1

for i in Range(N): n

for j in range(N): n^2

count += 2 2n^2

}

}

for k in range(N): n

count += 1 2n

}

4n + 3n^2 + 1

O(n^2)

**Question 8**:

Find the time complexity of following code.

def find\_duplicates(array): 1

for i in range(len(array)): n

for j in range (i + 1, len(array)): 2n^2

if array[i] == array[j]: 3n^2

print(array[j]) n^2

6n^2 + n + 1

O(n^2)

**Question 9**:

Find the time complexity of following code.

for i in range(n \* 1): n

for j in range(n + 2): n^2

for k in range(n \* 3): n^3

print(i, j, k) n^3

2n^3 + n^2 + n

O(n^3)

**Question 10**:

Lets’ say we have a polynomial 3x + 9x+8y=79. This function finds all the solutions that satisfy our equation. Find the time complexity of following code.

def find\_xyz(n): 1

solution = [] 1

for x in range(n): n

for y in range(n): n^2

for z in range(n): n^3

if (3\*x + 9\*y + 8\*z == 79): 6n^3

solution.append([x, y, z]) n^3

return solution 3

8n^3 + n^2 + n + 3

O(n^3)

**Question 11:**

Consider following codes and sample inputs one by one for experimental analysis of two sorting algorithms namely: Bubble Sort and Selection Sort.

0.09794

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Algorithms** | **Sorted Array** | | | **Unsorted Array** | | |
| **Best Case** | **Average Case** | **Worst Case** | **Best Case** | **Average Case** | **Worst Case** |
| Bubble Sort | 0.0156 | 0.0279 | 0.0312 | 0.1406 | 0.2031 | 0.3620 |
| Selection Sort | 0.04687 | 0.07813 | 0.10936 | 0.09374 | 0.09794 | 0.1109 |

|  |
| --- |
| **Sample Inputs** |
| sorted\_array = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26] |
| Unsorted\_array = [8, 22, 6, 10, 14, 2, 24, 4, 16, 26, 12, 18, 20] |

|  |  |
| --- | --- |
| **# Optimized Bubble sort**  def bubbleSort(array):    # loop through each element of array  for i in range(len(array)):    # keep track of swapping  swapped = False    # loop to compare array elements  for j in range(0, len(array) - i - 1):  # compare two adjacent elements  # change > to < to sort in descending order  if array[j] > array[j + 1]:  # swapping occurs if elements  # are not in the intended order  temp = array[j]  array[j] = array[j+1]  array[j+1] = temp  swapped = True    # no swapping means the array is already sorted  # so no need for further comparison  if not swapped:  break  data = []  bubbleSort(data)  print('Sorted Array in Ascending Order:')  print(data) | **# Selection sort**  def selectionSort(array, size):    for step in range(size):  min\_idx = step  for i in range(step + 1, size):    # to sort in descending order, change > to < in this line  # select the minimum element in each loop  if array[i] < array[min\_idx]:  min\_idx = i    # put min at the correct position  (array[step], array[min\_idx]) = (array[min\_idx], array[step])  data = []  size = len(data)  selectionSort(data, size)  print('Sorted Array in Ascending Order:')  print(data) |

|  |
| --- |
| **# For calculating time**  from time import time  start time = time( ) # record the starting time  run algorithm  end time = time( ) # record the ending time  elapsed = end time − start time # compute the elapsed time |

Now supply the same inputs to Linear Search and Binary Search Algorithm for experimental analysis and look for any number.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Algorithms** | **Sorted Array** | | | **Unsorted Array** | | |
| **Best Case** | **Average Case** | **Worst Case** | **Best Case** | **Average Case** | **Worst Case** |
| Linear Search | 0.0009 | 0.00100 | 0.0019 | 0.0312 | 0.0156 | 0.02498 |
| Binary Search | 0.00005 | 0.000099 | 0.000156 | 0.01599 | 0.01499 | 0.0259 |

|  |  |
| --- | --- |
| **# Linear Search**  def linearSearch(array, n, x):  # Going through array sequencially  for i in range(0, n):  if (array[i] == x):  return i  return -1  array = []  element = 0  n = len(array)  result = linearSearch(array, n, element)  if(result == -1):  print("Element not found")  else:  print("Element found at index: ", result) | **# Binary Search**  def binarySearch(array, x, low, high):  # Repeat until the pointers low and high meet each other  while low <= high:  mid = low + (high - low)//2  if array[mid] == x:  return mid  elif array[mid] < x:  low = mid + 1  else:  high = mid - 1  return -1  array = []  element = 0  result = binarySearch(array, element, 0, len(array)-1)  if result != -1:  print("Element is present at index " + str(result))  else:  print("Not found") |