Design & Analysis: Algorithms Homework 1

Taaruni Ananya | CSC 4520

Problem 1. (20 points) For each of the following pairs of functions, indicate whether the first function of each of the following pairs has a lower, same, or higher order of growth (to within a constant multiple) than the second function.

a. $n(n + 1)$ and $2000n^2$
   $n(n + 1) = n^2 + n = \Theta(n^2)$
   $2000n^2 = \Theta(n^2)$
   Both functions have the same order of growth with $\Theta(n^2)$.
b. $100n^2$ and $0.01n^3$
   n^2 / n^3, the second function grows faster than the first function.
c. $\log_2 n$ and $\ln n$
   Using change of base formula, $\log_2 n = \log(n) / \log(2) = \ln(n) / \ln(2) =$ **$\ln(n)$** $(1 / \ln(2))$. Both functions have the same order of growth with $\Theta(\ln(n))$.
d. $\log^2 n$ and $\log_2 n^2$
   $\log^2 n = (\log n)^2$ and $\log_2 n^2 = 2\log_2 n = \log n$. Both functions have the same order of growth with $\Theta(\log n)$.
e. 2^{n-1} and 2^n
   $2^n / 2^1 = 2^n$. Both functions have the same order of growth with $\Theta(2^n)$

Problem 2. (20 points) List the following functions according to their order of growth from the lowest to the highest:

$$(n - 2)!,\ 5\log(n+100)^{10},\ 2^{2n},\ 0.001n^4 + 3n^3 + 1,\ \ln^2 n,\ \sqrt[3]{n},\ 3^n$$

$(n - 2)! = n! = \Theta(n!)$
$5\log(n+100)^{10} = \Theta(\log n)$
$2^{2n} = (2^2)^n = \Theta(4^n)$
$0.001n^4 + 3n^3 + 1 = 0.001n^4 = n^4 = \Theta(n^4)$
$\ln^2 n = (\ln n)^2 = \Theta((\ln n)^2)$
$\sqrt[3]{n} = n^{1/3} = \Theta((n^{1/3})$
$3^n = \Theta(3^n)$

In order from lowest to highest:

1. $5\log(n+100)^{10} = \Theta(\log n)$
2. $\ln^2 n = (\ln n)^2 = \Theta((\ln n)^2)$
3. $\sqrt[3]{n} = n^{1/3} = \Theta((n^{1/3})$
4. $0.001n^4 + 3n^3 + 1 = 0.001n^4 = n^4 = \Theta(n^4)$
5. $3^n = \Theta(3^n)$
6. $2^{2n} = (2^2)^n = \Theta(4^n)$
7. $(n-2)! = n! = \Theta(n!)$

Problem 3. (30 points) Consider the following recursive algorithm for computing the sum of the first n cubes: $S(n) = 1^3 + 2^3 + \ldots + n^3$

ALGORITHM S (n)
//Input: A positive integer n
//Output: The sum of the first n cubes
if n = 1 return 1
else return S(n − 1) + n * n * n

Set up and solve a recurrence relation for the number of times the algorithm's basic operation is executed.

Recurrence relation:

$S(n-1) + 1, S(1) = 1$

$S(n-1) = S(n-2) + (n-1) + 1 + 1 = S(n-2) + 2$

$S(n-2) = S(n-3) + (n-2) + (n-1) + 1 + 1 + 1 = S(n-3) + 3$

$S(n) = S(n-i) + (n-i+1) + (n-i+2) + (n-i+3) + \ldots\ldots + 1$

$1 = n - i = i = n - 1$

$\Theta(n)$

Problem 4. (30 points) Develop a program in Python to implement Bubble Sort, Selection Sort, and Insertion Sort algorithms. Execute these algorithms on input arrays that are sorted, unsorted, and reverse sorted, with sizes of 50, 100, 200, 400, 500, and 1000 elements. Additionally, generate plots visualizing the sorting performance results, similar to the reference figures shown below.

```
import random
import time
import matplotlib.pyplot as plt
```

```python
# Bubble sort implementation
def bubble_sort(arr):
    length = len(arr)
    for x in range(length):
        for y in range(0, length - x - 1):
            if arr[y] > arr[y+1]:
                arr[y], arr[y+1] = arr[y+1], arr[y]

# Selection Sort implementation
def selection_sort(input):
    n = len(input)
    for x in range(n - 1):
        min_index = x
        for y in range(x+1, n):
            if input[y] < input[min_index]:
                min_index = y
        input[x], input[min_index] = input[min_index], input[x]

# Insertion sort implementation
def insertion_sort(input):
    for x in range(1, len(input)):
        key = input[x]
        y = x - 1

        while y >= 0 and key < input[y]:
            input[y+1] = input[y]
            y -= 1
        input[y+1] = key

# Function to run algorithms, based on the different types of
order
```

```
def run_algorithms(lists, order_type):
   results = {"Bubble Sort": [], "Selection Sort": [],
"Insertion Sort": []}

   for size in lists:
       if order_type.lower() == "sorted":
           array = list(range(size)) # works with a sorted
array
       elif order_type.lower() == "reverse sorted":
           array = list(range(size, 0, -1)) # works in reverse
for reverse sorted
       else:
           array = [random.randint(0, size*10) for _ in
range(size)] # works for random/unsorted

       array_copy = array.copy()
       start = time.perf_counter()
       bubble_sort(array_copy)
       results["Bubble Sort"].append(time.perf_counter() -
start) # performs the results for bubble sort

       array_copy = array.copy()
       start = time.perf_counter()
       selection_sort(array_copy)
       results["Selection Sort"].append(time.perf_counter() -
start) #  performs the results for selection sort

       array_copy = array.copy()
       start = time.perf_counter()
       insertion_sort(array_copy)
       results["Insertion Sort"].append(time.perf_counter() -
```

```
start) # performs the results for insertion sort

    return results

# the following will input the necessary types and sizes
sizes = [50, 100, 200, 400, 500, 1000]
order_types = ["sorted", "unsorted", "reverse sorted"]

# the following is for plotting the different graphs using
Matplotlib
for order in order_types:
    results = run_algorithms(sizes, order)
    plt.figure(figsize=(10,6))
    for algorithm in results:
        plt.plot(sizes, results[algorithm], marker='o',
label=algorithm)
    plt.title(f"Order of Growth - {order} Data")
    plt.xlabel("Input Size")
    plt.ylabel("Time Taken (s)")
    plt.xlim(50, 1000)
    plt.legend()
    plt.grid(True)
    plt.show()
```