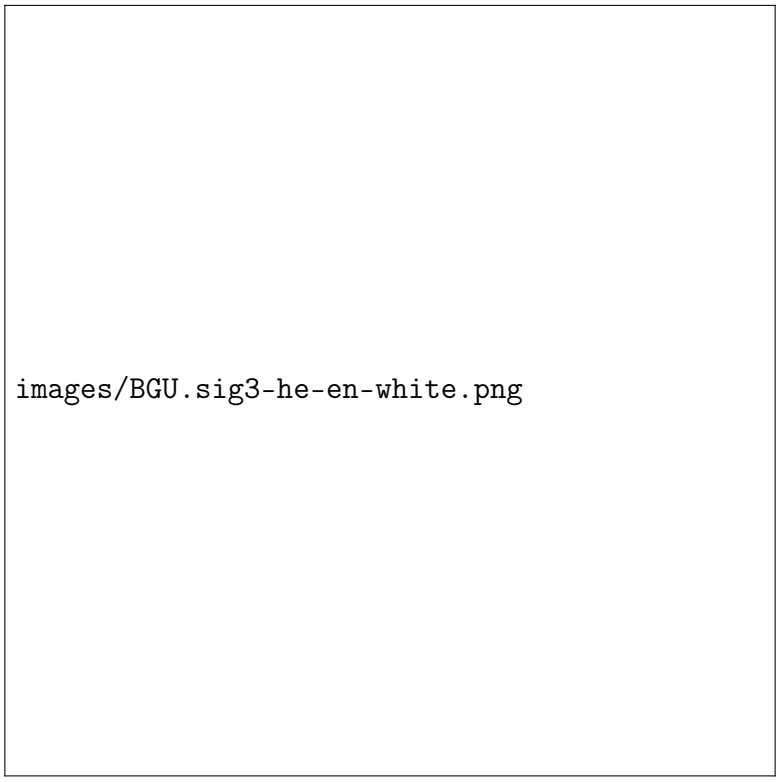
Ben-Gurion University of the Negev

Data Structures
202.1.1031

Assignment No. 5

Responsible staff members:
Dr. Or Sattath Ilya Kaufman
Publish date: June 8th, 2023
Submission date: June 22nd, 2023


images/BGU.sig3-he-en-white.png

0 Integrity Statement

I, <Aseel Biadsi (213758758), Hamza Mjadly(325043552)>, certify that the work I have submitted is entirely my own. I have not received any part of it from any other person, nor have I given any part of it to others for their use.
I have not copied any part of my answers from any other source, and what I submit is my own creation. I understand that formal proceedings will be taken against me before the BGU Disciplinary Committee if there is any suspicion that my work contains code/answers that is not my own.
If you have relied on or used an external source, you must cite that source at the end of your integrity statement. External sources include shared file drivers, Large Language Models (LLMs) including ChatGPT, forums, websites, books, etc.

Homework Guidelines

- The submission is in pairs or individuals. We recommend working in pairs to encourage discussion and mutual inspiration. To avoid doubt, if the assignment was submitted in pairs, all submitted items should be the outcome of joint work.
- You must define a group in the course Moodle site and assign yourself to it, even if you choose to submit individually. you can visit the https://moodle.bgu.ac.il/moodle/mod/choicegroup/view.php?id=2448687 group assignment page for more info.
- Your answers must be typed and submitted as a pdf file. We recommend using the LaTeXtemplate and submitting its compilation result, But you can also use Word or any other text editor you prefer. Your file's name should be identical to your assigned group name in Moodle. For instance, *Assignment*5_*p*_100.*pdf* or *Assignment*5_*s*_20.*pdf*
- Questions regarding the assignment should be asked in the https://moodle.bgu.ac.il/moodle/mod/forum/view.php?id=2448695 dedicated forum in Moodle or during the office hours of the responsible staff members. The forum is intended for a discussion between the students regarding the assignment. The assignment staff will answer questions in the forum only if clarification is required. Clarifications, if needed, will be published once every 24 hours.
- You don't need to prove or elaborate on things learned in class but you should prove any claim that wasn't shown in lectures or practical sessions.

1 Huffman Code - Encoding Algorithm

We would like to improve the run time complexity of the Huffman encoding algorithm seen in class, given that the input frequencies are sorted. More formally:
Input: An array of size σ such that each entry consists of a pair (c, f_c), where c is a character and f_c is the frequency of the character c.
Output: A binary node representing the root of the prefix code (as in the algorithm learned in class).
Instructions: You are allowed to use only two queues (which are not priority queues) without any additional data structures. Assume that a queue Q has an operation $peek(Q)$ that returns the element at the top of the queue (without removing it from the queue) in $\Theta(1)$.

1. Write a pseudo-code for an algorithm that follows the given instructions.
2. Analyze the run time complexity of the proposed algorithm (make sure that your algorithm improves the asymptotic run time complexity of the algorithm that was learned in class).

Solution:

Algorithm 1 Pseudo Code

Input :

```
Size ← |c|
Q1 ← queue of C
Q2 ← queue
for i = 1 to Size − 1 do
    Allocate a new node z
    if Q1 is empty then
        z.left ← Q2.dequeue
    else if Q2 is empty then
        z.left ← Q1.dequeue
    else
        if Q2.peek.fc ≤ Q1.peek.fc then
            z.left ← Q2.dequeue
        else
            z.left ← Q1.dequeue
    Do the same with z.right
    Q2.enqueue (z)
```

Output: Q2.dequeue

1.

2. The run-time complexity is: $\boxed{\theta(C)}$.

   Analysis (at most 3 lines): The time complexity of the given algorithm is $\Theta(C)$ because it iterates through the loop $C-1$ times, where C is the size of the queue. Each iteration performs constant-time operations, resulting in a linear time complexity proportional to the size of the queue.

2 Lempel-Ziv 78 - Decoding Algorithm

Let s be a string over the alphabet $\{a, b, \ldots z, \_\}$. The output of the $LZ78$ compression algorithm on s produced the following pairs:

$$(0,i),(0,\_),(0,l),(0,o),(0,v),(0,e),(2,d),(0,a),(0,t),(8,\_),(0,s),(9,r),(0,u),(0,c),(9,u),(0,r),(6,s)$$

Decode s.
Solution: $\boxed{s = \text{i love data structures}}$

3 Amortized Analysis

We define the following function:

Algorithm 2 Function `f()`

```
Function f ():
    i ← i + 1
    if i = 2^k for some k ∈ N then
        j ← 0
        while j < i do
            j ← j + 1
        end
    end
```

where i is a global variable initiated with 0. What is the amortized cost of a single $f()$ operation?

Solution: The amortized cost of a single f operation is: $\boxed{\theta(1)}$.

Analysis: At first, we will divide the function $f()$ into two types: low cost and high cost.

High cost: It occurs $\log(n)$ times and costs i (express the i-th call to the function) $(2, 4, 8, \ldots, n)$. We can calculate the sum of a geometric series using the formula $s_n = \frac{a_1 \cdot (q^n - 1)}{q-1}$. The total cost is $2n - 2$.

Low cost: The high cost occurs $\log(n)$ times, which means that the low cost occurs $n - \log(n)$ times. The cost of each low-cost operation is 1, resulting in a total cost of $n - \log(n)$.

Total: Using the accounting method, a series of n calls to the function will cost $3n - 2 - \log(n)$. Therefore, the average cost is $\frac{3n-2-\log(n)}{n}$, which is less than 3. Thus, the average cost is $\Theta(1)$.

4 Sorting

Given an array of numbers A, describe an algorithm that returns the $\lfloor \frac{n}{\log n} \rfloor$ smallest numbers that appear exactly once, sorted by their value. You can assume that there are at least $\lfloor \frac{n}{\log n} \rfloor$ unique values in A. The algorithm should have an expected runtime complexity of $\Theta(n)$ and use at most $\Theta(n)$ extra space.
Example:
Input: $5, 7, 31, 9, 9, 23, 1, 5, 7$
Output: $1, 23$

Algorithm: At first, we build a hash table of size n using chaining and we insert into the hash table the keys and we initialize a frequency filed. we will go over the array and for every key we will search for it in the hash table if its exist we increase the field frequency by 1 , if its not we add to the the hash table,
now we have the keys and the frequencies, we build an auxiliary array that we will insert to it every key that the field frequency is 1.
now we have the keys that appears exactly one, there is no need to sort them all because we don't know how many they are
now we can use the algorithm select that we have learned in the lecture in order to find the $\lfloor \frac{n}{\log n} \rfloor$ index. after we find it we build the output array of size $\lfloor \frac{n}{\log n} \rfloor$ and we add to it every element in the auxiliary array with key smaller than the key we got from the function select. now we have an array with $\lfloor \frac{n}{\log n} \rfloor$ elements which can be sorted using merge sort . and we return this array

Now, we create a new array of size $\lfloor \frac{n}{\log n} \rfloor$. We scan our sorted array using the following method:

- If the immediately preceding or immediately following member is equal to the current member, we do not insert it into the output array. - Otherwise, we insert the member into the first available position in the output array (using an index variable) and multiply it by M to retrieve its true value. - We stop when the output array is filled.

Time Complexity:
1. initializing a the hash table $O(n)$ time.
2. insert the elements into the hash table and increase the key field when needed also costs $O(n)$ expected.
3. adding the elements that appear one time to the auxiliary array costs $O(n)$ time.
4. calling of select function costs $O(n)$ time. 5. Creating the output array and inserting the elements as described above also requires $O(n)$ time. 6. sort the output array using merge sort cost (in this situation) $O(n)$

Therefore, the overall running time of the algorithm is $O(n)$ expected.

Memory usage analysis: We create two arrays, one of length n and the other of length $\lfloor \frac{n}{\log n} \rfloor$.
The bucket sort algorithm also uses $O(n)$ memory.

Therefore, the total memory usage is $O(n)$.

5 Sorting

Given an array A with $n - 1$ numbers where $n = 2^k$ for some integer k. One of the values appears exactly $\frac{n}{2}$, another appears exactly $\frac{n}{4}$ and so on. More formally, for all $1 \leq k \leq \log n$ there exists a unique value that appears exactly $\frac{n}{2^k}$ times. Describe an algorithm that sorts A with a run time complexity of $\Theta(n)$, and analyze its running time.

Example:
Input: $5, 1, 2, 5, 2, 5, 5$
Output: $1, 2, 2, 5, 5, 5, 5$

algorithm: to solve this question we will use the select function that we have learned in the lecture because the length of the array a is n-1 we can deduce from the pigeon hole principle that if we use the function select and we gave it to return the value of the index n/2 in the array it will return the integer that appears exactly n/2 times, after that we can put this integer into an array (of size log n) with satellite-data,(we will call it frequencies) then we can copy a new array with n/2 elements that are different from the integer we got. now we can do the same steps that we did before recursively (find the middle element and copy it to an array...) now we have an array with log n elements we can sort it which cost less than O(n)...

now we initialize the output array and we inter the elements in sorted way (with the help of the sorted array of the size log n) and every element we add it to the array frequency times

this is the array in sorted way

Time Complexity: the function sort costs at most 10n (we proves that in the lecture), therefor the select run time series is (10n, 5n, ..., 1), the sum of this series is 20n, which is still O(n), sorting an array with length log n costs less than O(n) (if we used insertion sort), to copy the element to the output array costs O(n). in total the algorithm costs O(n).

6 Sorting

Given an array A with n elements such that each element contains a key (an integer) and additional satellite data. The array contains exactly k unique key values.

1. Assume that $k \in \Theta(\log n)$. Describe a stable sorting algorithm that sorts A with a run time complexity of $\Theta(n \log \log n)$

   Algorithm: we begin by initializing an AVL tree, this tree guarantees that members with the same key are not inserted multiple times.

   Next, we traverse the input array and insert each key into the AVL tree, by creating new node with two fields key and $frequency$, if the node key does not exist in the tree we will add it with $frequency = 1$, else we will add 1 to the already existing node $frequency$. By adhering to the properties of the AVL tree, only a maximum of k members with unique special keys will be included, resulting in a balanced tree with a height of approximately log k.

   After that we do in-order scan on the tree and we update every node $frequency$ by the the equation $this.frequency+ = prev.frequency$, by that we know every range of key in the output array (its $frequency-$ prev node $frequency$.

   Now, we initialize an output array of size n, to retrieve the elements in their original order, we perform scan to the input array from the last index to the first, to every key we search it in the tree, then add it to the output array in index $frequency$ that we founded in the node.

   In this way, we effectively sort the input array, maintain order, and produce the desired output array.

   Time Complexity: The algorithm's time complexity is $\theta(nloglogn)$ when k (the number of elements inserted into the AVL tree) equals log n, where n is the size of the input array. This complexity arises from the following key steps:

   Initializing the AVL tree: $\theta(1)$ time complexity.

   Scanning and inserting keys into the AVL tree: $\theta(nloglogn)$ time complexity.

   Initializing and inserting the elements to the output array: $\theta n \log \log n$ time complexity.

   In summary, the algorithm efficiently sorts the input array, maintains order, resulting in a time complexity of $\theta(nloglogn)$ for the given scenario.

2. Assume that $k \in \Theta(\sqrt{n})$. Describe a stable sorting algorithm that sorts A with an expected run time complexity of $\Theta(n)$

   Algorithm: At first we build a hash table (chaining) with size k that takes an integer, we insert into the hash table a key and inside the node that that hold the key we initials field with frequency to be 1 (if it

doesn't exist), else (if the hash table contain s the key) we add 1 to the frequency, we do that for every element in the input array, after that we sort the keys that in the hash table array using insertion sort .

then we can build an auxiliary array just like that in the counting sort(which holds all the Numbers that are smaller than the index value (in our case the key) in the array) after we have this array we can do the same steps that the counting sort did to sort the input array, which give us a stable sorting.

time-complexity: to initialize an hash table cost O(n) to insert the keys into the hash table and increase the frequency (when needed) cost O(n) expected

to sort the keys using insertion sort cost O(n) to build the auxiliary array that I have mentioned above cost also O(n).

to insert the values just into the output array (like counting sort) cost O(n)

which means that the run time complexity is :O(n) expected

You may use $\Theta(n)$ extra space.
Example:
Input: $(5,a),(1,a),(2,a),(5,d),(2,a),(5,c),(5,b)$
Output: $(1,a),(2,a),(2,b),(5,a),(5,d),(5,c),(5,b)$

7 Graphs

Definition: Given two vertices v, e the shortest even edge path, is the shortest path from v to u that has an even number of edges.
Given a connected graph $G = (V, E)$ represented using adjacency lists and a vertex $s \in V$, describe an algorithm that finds the length of the shortest even edge path from each $v \in V$ to s (if no such path exists the length is $-\infty$) with a run time complexity of $\Theta(V + E)$ and analyze its run time complexity.

Instructions: Use a bipartite graph $G' = (V', E')$, where $V' = L \cup R$. $L = \{v^l | v \in V\}$, $R = \{v^r | v \in V\}$, $E' = \left\{ \left(v^l, u^r\right), \left(u^l, v^r\right) \mid (v,u) \in E \right\}$

Algorithm: at first we build the G' graph as shown in the instruction (a bipartite graph), we can notice that the distance between every vertice in the left side and any other vertice in the right side is odd, therefore if we run the BFS algorithm in left vertice of s (the input vertice)for each vertice V, V^l express the shortest even distance between S and V and so the right side but in symmetric way, so we can deduce that the shortest odd distance between S and V is the distance between S^l and V^r that the BFS return , so the shortest even distance between S and $v_1, v_2, ..., v_n$ is the shortest distance between S and $v_1^r, v_2^r, ..., v_n^r$.

time-complexity: to build the G' graph cost O(V+E). the BFS algorithm cost O(V+E) to insert the distances to the output array cost O(V). in total: the time complexity is : O(V+E).

Good Luck!