HOMEWORK 9

ELSY FERNANDES

1001602253

CSE 2320 - Homework 9

NAME: ELSY FERNANDES

Total points: 100 Topics: Graph adjacency matrix representation, connected components, hash tables, quicksort, radix sort, count sort, Timsort.

P1 (40 pts) Assume a graph represents a social network: the vertices are people and the edges are friendship relation between them. Implement a program that does the following:

- reads the graph with input redirection.
- prints the graph to verify it was loaded correctly.
- find the connected components and label them with numbers (starting at 1) and
- print the connected components. In particular print the names of the people in that connected component.

You can assume the graph is an undirected graph.

The graph data is given in the order:

- N (number of vertices)
- Vertices: N lines each containing one string with a vertex name.
- Edges: A number of lines in format: "name1 name 2". The line "-1 -1" indicates the end of lines with edges.

Files: data1.txt, run1.txt.

Save the program in a file called **graph.c**.

Hint: you can adapt the DFS (Depth First Search) algorithm to label connected components.

P2. (7 points) Is Quick_Sort (as given in CLRS, page 171) stable?

If yes, prove it. If no, give an example array, A, (however small or big), sort it with Quick_Sort, and show what the algorithm does that makes it not stable. Use the original array and the final, sorted array to base your proof (do not base your proof on a partially sorted array).

Hint: Focus on the pivot jump. Quick Sort is NOT STABLE. It swaps the Elements. When in array 2 Elements ar Equal, after multiple Sort we are not guaranted to main the order of those two Elements those are Equal.

For eg:-> A: 4, 2, 1, 4, 3 3 is a pivot, lets Mark 4 as 4 First and 4 second to understand it better

original A: 4[First], 2, 1, 4[Second], 3

: 2, 4[First], 1, 4 [Second], 3

: 2, 1, 4[first], 4[second], 3

: 2, 1, 3, 4[Second], 4[First] ->Here as we can see in the sorted array the position of ->4[second] is Before than 4[First] as compared to its original Array.

P3. (7 points) Given the array A = < 8, 6, 9, 2, 7, 1, 5, 10, 6 >, using the Quicksort Lecture or Figure 7.1, CLRS page 172, as a model, show **the execution** of the Partition function. Show the <= partition by **circling the last element of it** and show the > partition by putting **a square around the last element of it.**

	0	1	2	3	4	5	6	7	8
Original array:	8	6	9	2	7	1	5	10	6
	$\boxed{8}$	6	9	2	7	1	5	10	6
	6	$\boxed{8}$	9	2	7	1	5	10	6
	6	8	$\boxed{9}$	2	7	1	5	10	6
	6	2	9	$\boxed{8}$	7	1	5	10	6
	6	2	9	8	$\boxed{7}$	1	5	10	6
	6	2	1	8	7	$\boxed{9}$	5	10	6
	6	2	1	5	7	9	$\boxed{8}$	10	6
	6	2	1	5	7	9	8	$\boxed{10}$	6
	6	2	1	5	6	9	8	10	$\boxed{7}$

Return 4

P4. (9 points) (Radix sort)
Show how **LSD radix sort** sorts the following numbers in the given representation (base 10). Show the numbers after each complete round of count sort.

Index:	0	1	2	3	4	5	6
Original Array:	513	145	320	235	141	433	2
1st pass	320	141	002	513	433	145	235
2nd pass	002	513	320	433	235	141	145
3rd pass	002	141	145	235	320	433	513

count → 1st pass

0 1 2 3 4 5
1 1 1 2 0 2
1 2 3 5 5 7 → cum sum

count → 2nd pass

0 1 2 3 4 5
1 1 1 2 2 0
1 2 3 5 7 7 → cum sum

count → 3rd pass

0 1 2 3 4 5
1 2 1 1 1 1
1 3 4 5 6 7 → cum sum

P5. (10 points) **Count sort, radix sort**

Estimate the performance of count sort and radix sort for the following problems and set-ups. Assume you are dealing with numbers on b=32 bits. Only positive values are allowed (so there is no sign bit, all 32 bits are used to represent the positive value). Let:

- b – number of bits
- k - number of different possible values of keys
- r – number of bits used for a specific radix
- N – size of array to be sorted.
- d – number of digits in a certain base (on r-bits) representation.

Continue to fill-in the table. Express every term as a power of 2. In the Θ use the values for the terms. E.g. N^2 for N=8 would be replaced with 2^6(=64). In order to compare quantities easily, we express everything as a power of 2 in the table below (see $7 \approx 2^{2.8}$). See clarification.

N	Count sort			Radix sort for r = 5 bits				Optimal radix sort (with optimal r)					Best method	
N	k	Θ	Dominant term	k	d	Θ	Dominant term	r	k	d	Θ	Dominant term	Smallest dominant term	Method
$8=2^3$	2^{32}	2^3+2^{32}	2^{32}	2^5	$7 \approx 2^{2.8}$	$2^{2.8}(2^3+2^5)$	$2^{7.8}$	2	2^2	$16=2^4$	$2^4(2^3+2^2)$	2^7	2^7	optimal radix
$10^{15} \approx 2^{50}$	2^{32}	$2^{50}+2^{32}$	2^{50}	2^5	$7=2^{2.8}$	$2^{2.8}(2^{50}+2^5)$	$2^{52.8}$	49	2^{49}	$0.65 \cong 2^{0.1}$	$2^{0.1}(2^{50}+2^{49})$	$2^{50.1}$	$2^{50.1}$	Count sort

P6. (10 points) **self study of Timsort** Read this article, and answer the Timsort questions below based on it.

(2pt) Timsort was created by:..Tim....Peters........ It is used as the default sorting algorithm for: Python, Java, the Android, GNU octave

(3pts) Time complexity: Best case: Ω(...n.......) Average case: Θ(.....n log(n)..) Worst case: O(....n log(n)..)

(1pts) What two sorting algorithms does it combine? ..Insertion Sort.....and...Mergesort

(2 pt) Circle your answer: Is it stable? YES / No Does it do well on arrays with preexisting structure? YES / No

(1 pt) What data does "~ sort" indicate in Tim Peters's introduction to Timsort found here? ..Already ordered Element. Natural Runs

(1pt) This Wikipedia article discusses a bug found in Timsort. What Java error does that bug produce? ..If the Stack Size increased, then it wouldn't provide a sufficient checks. Java error was it avoided a overflow checks because implementation preallocated a stack sufficient to sort 2^{64} bytes

P6. (5 points) **Hashing**

The images below show the occupancy of two hash tables. Both tables have **the same size**, the **same items** hashed in them, and both use **open addressing**. However they differ in the way they find an available slot in the table. Which one is a better hash table and why? (You do NOT have to deduce how they find the next available slot. You simply have to judge which one would behave better based on this image.)

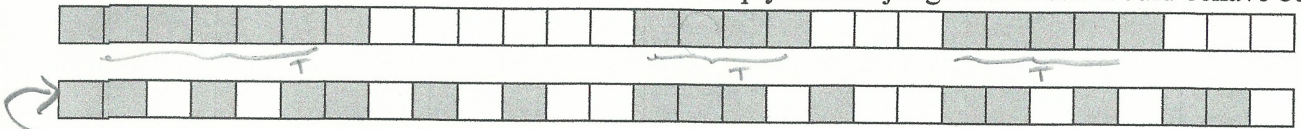

Table 1

Table 2

→ Table 2 is a better has table and Efficient one. We have a clustered data in table 1. In table 1 we see clustered Blocks, Bigger the T (as shown in Table 1) Bigger the probability of collision. and also it takes time to search for the free slots as these are forming the clusters inside the hash table which makes search time increase. where as table 2 uses other method of open addressing in order to avoid collision which finds free slot in less time compared to table 1

P7. (12 points) **Hashing**

a) (5 points) Give an example of a bad hash function for strings (that generates many collisions). Justify why it is bad: find some strings that will hash to the same cell. Suppose we have 4 strings, cat, Horse, dog, act

lets say we have hashed these strings and found key:→

Cat → 3
Horse → 5
dog → 3
act → 3

Here all 3 strings are hashed to same location. Hence it is a Bad Hash function

So it always better to hash strings based on their Ascii value with good Hash function.

→ Some strings has same hash value are Sauce & Cause, Fry and hut,

b) (7 points) In the hash table below * and + indicate the cells probed when trying to insert two different items. These items are originally hashed to the same slot (see +* at index 5). The star, *, shows the probed cells for item and the plus, +, shows the probed cells for the other item. You can assume that the table size is very big (e.g. more than 1000) and the slots shown did not require a mod (%) operation (that is we did not have to wrap around).

1) What type of open addressing was used? Justify.

Here we use Double Hashing. Say suppose a string "blahb" has key 5. But if key 5 is already occupied. it is hashed again (like it has a different hash function) if that produces value 3 then it will have a Jump of 3. If this Slot is taken, it will look for a new slot with a Jump of 3 again.

2) Give the next slots to be checked for each item (show where the next * and where the next + will be).

like in the Below table;→

Index	
…	…
5	+ *
…	…
8	+
9	*
…	…
11	+
12	
13	*
14	+
…	…
17	* +
…	…

INDEX	
20	*

Next + will be at 17

Next * will be at 20

Remember to include your name at the top.

Write your answers in this document or a new document called **2320_H9.pdf**. It can be hand-written and scanned, but it must be uploaded electronically. Place **2320_H9.pdf** and **graph.c** in a folder called **hw9**, zip that and send it.