CSE 2320 - Homework 9

NAME:____

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

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.

No. It is not stable. Example 1,2,6a,6b,5 after partition by 5: 1, 2, 5, 6a, 6b.

Quicksort will be called for <1,2> and <6b, 6a>, but it will not move any element (the swap will keep the pivot in the same place).

An even shorter example is: <1,6a,6b> after partition, because of the pivot swap we get: <1,6b,6a> and the algorithm ends (<1> and <6a> are base cases).

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**
	8								
	6	**8**							
	6	**8**	**9**						
	6	**2**	**9**	**8**					
	6	**2**	**9**	**8**	**7**				
	6	**2**	**1**	**8**	**7**	**9**			
	6	**2**	**1**	**5**	**7**	**9**	**8**		
	6	**2**	**1**	**5**	**7**	**9**	**8**	**10**	
	6	**2**	**1**	**5**	**6**	**9**	**8**	**10**	**7**

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
	320	**141**	**2**	**513**	**433**	**145**	**235**
	2	**513**	**320**	**433**	**235**	**141**	**145**
	2	**141**	**145**	**235**	**320**	**433**	**513**

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) **r = min(b, lg(N))**					Best method	
N	k	Θ	Dominant term	k	d	Θ	Dominant term	r	k	d = ceil(b/r)	Θ	Dominant term	Smallest dominant term	Method
$8=2^3$	2^{32}	2^3+2^{32}	2^{32}	2^5	$7\approx 2^{2.8}$	$2^{2.8}(2^3+\mathbf{2^5})$	$\mathbf{2^{7.8}}$	3	2^3	$11\approx 2^{3.5}$	$2^{3.5}*(2^3+2^3)$	$2^{6.5}$	$2^{6.5}$	Optimal radix
$10^{15}\approx 2^{50}$	2^{32}	$2^{50}+2^{32}$	2^{50}	2^5	$7\approx 2^{2.8}$	$2^{2.8}(2^{50}+2^5)$	$2^{52.8}$	32	2^{32}	1	$1*(2^{32}+2^{50})$	2^{50}	2^{50}	Count & opt radix

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 platform, GNU Octave

(3pts) Time complexity: Best case: Ω(…N….) Average case: Θ(…NlgN…..) Worst case: O(…NlgN …)

(1pts) What two sorting algorithms does it combine? ……merge sort & insertion sort …….

(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? …data with large masses of equal elements...

(1pt) This Wikipedia article discusses a bug found in Timsort. What Java error does that bug produce? …array out of bound……………………..

P7. (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.)

▨	▨	▨	▨	▨	▨	▨							▨	▨	▨	▨				▨	▨	▨	▨	▨			

▨	▨		▨		▨	▨		▨		▨			▨	▨	▨		▨			▨	▨		▨		▨	▨	

The second one should have better performance because it has shorter sequences of consecutive occupied slots (shorter clusters). Another way to think of it is that the occupancy pattern in the second one seems more random (than that in the first one) and so the hashing seems to be better.

P8. (12 points) **Hashing**

a) (7 points) In the hash table below, two items are originally hashed to the same slot. The slots examined for inserting one of them are shown by a star and the slots examined for inserting the other are shown by a plus. 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.

Double hashing: the + and * are first hashed in the same cell, but next they move with different steps: step 3 for +, and step 4 for *.
(If it was quadratic, they should have both probed exactly the same cells).

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

Index	
…	…
5	+ *
…	…
8	+
9	*
…	…

11	+
12	
13	*
14	+
…	…
17	*
…	…

Next slot for +: 17 (=14+3)

Next slot for *: 21 (=17+4)

b) (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.

Hash(s) = length(s). All strings of the same length will hash to the same value. E.g.: cat, rat, see, top

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.