CSE 2320 - Homework 5 NAME: Tu Mai

Total points: 100 Topics: quick sort (with median of 3), merge sort, radix sort,, bucketsort, count sort, Timsort.

P1 (50 pts) Given files: data1.txt, run1.txt .

Implement Quick_Sort version discussed in class with the median-of-three improvement for the Partition function. Write the code in a file called *median3_quick_sort.c* .

a) When there is an even number of elements, round down when computing the middle index. For example for [8, 2, 5, *1*, 6, 9, 4, 0] it should use index 3 (because (0+7)/2 rounded down is 3. So it will find the median of array elements 8, 1, 0 (NOT 8, 6, 0).

b) (8 pts) When your program is run using **input redirection** from file data1.txt it should work fine without any changes to the file. E.g. sample run: `./a.out < data1.txt`

It should repeatedly read arrays by getting first the number of elements in the array, N, and then the actual elements (on a new line). It should stop when N is 0 or less. (To read one array, first read N, then use a loop to read N integers and save them directly in the corresponding position in the array.)

The file format will be:

```
N
Elements separated by spaces
N
Elements separated by spaces
...
```

E.g. the file below gives the arrays [2,1,7,9] and [8, 6, 9, 2, 7, 1, 5, 0, 6]

```
4
2 1 7 9
9
8 6 9 2 7 1 5 0 6
-1
```

Your program must work with this EXACT file format.

c) (8 pts) There is no upper bound on N so you must use dynamic memory for the array A. You should allocate memory for A, use A and then free it for every new array you read in during a single program run.

d) In order to make it easy to trace the method, in the Partition function print the following:

1. The array section (as given when the partition function starts).
2. The 3 elements that are being considered for the pivot (write these to the right of what is printed in part a).
3. The array section **after** the median was placed in the last position of the section.
4. The array section after the elements were moved around and the pivot is in its final place (the array section right before the Partition method finished).

e) Use the format below when printing. For 1), 3), 4) above, print only the section being processed, BUT show it aligned. Use formatted printing and use 4 spaces for each element. Leave empty spaces where there are no elements. See sample run file.

f) (10 pts) The program must produce the exact same output as mine.

g) (15 pts) The program should not have any memory errors when ran with Valgrind.

h) (9 pts) You must implement the Quicksort and Partition method covered in class (not any other version).

i)

P2. (7 points) Is Quick_Sort (as given in the class notes) 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

Example $1 \quad 8_A \quad 4 \quad 5 \searrow$ pivot $8_B \quad 2_A \quad 2_B \quad 7$

Aftersort $1 \quad 2_B \quad 2_A \quad 4 \quad 5 \searrow$ pivot $7 \quad 8_B \quad 8_A$

the keys of occurence of 3, 7 was dissordered after sorted

P3. (8 points) Assume merge_sort base case:

```
if ((right-left+1)<=15) {
   insertion_sort(A,left,right);
   return;
}
```

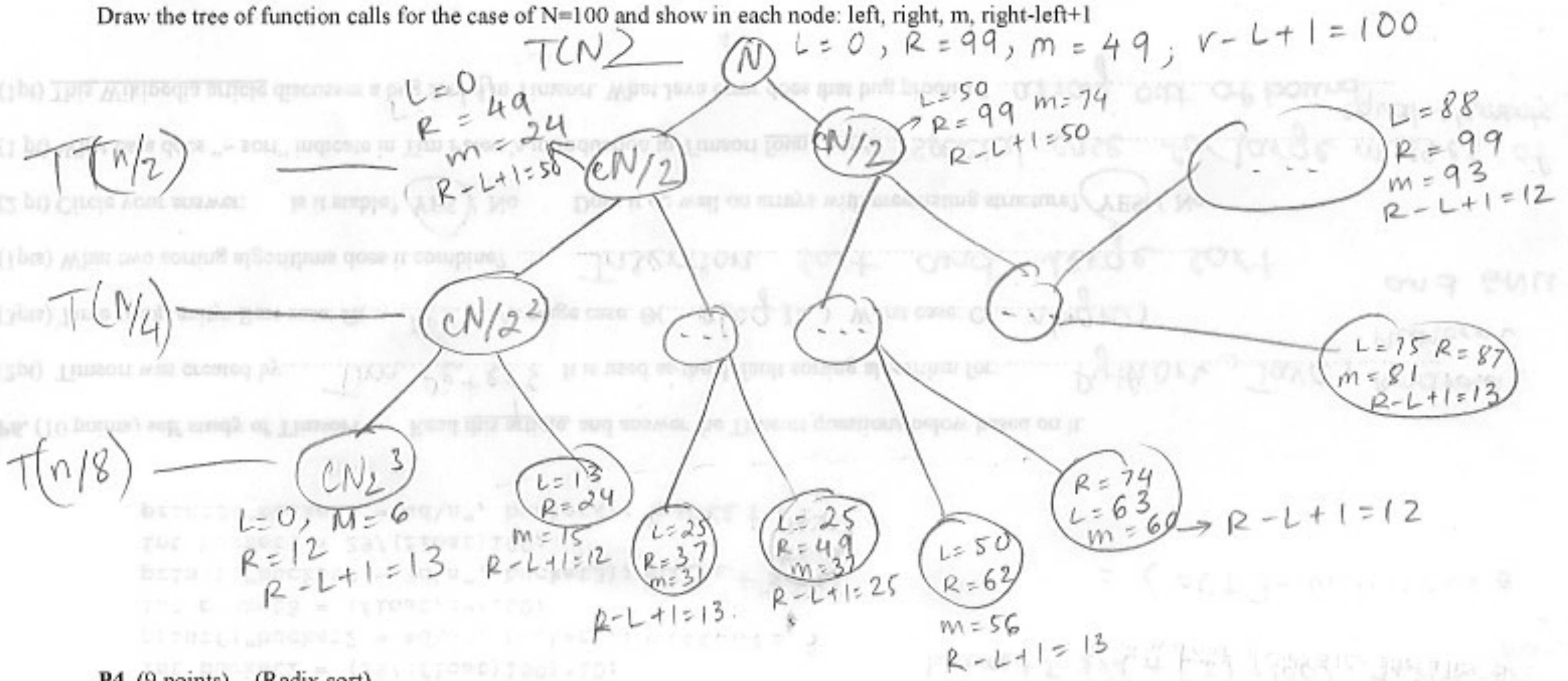
Compute the last level, k, (with leaves) of the tree produces by a call to merge_sort to sort an array, A, with N items. k = …….

What is the exact value for k if N is 100? k = …… 100/2

50/2 2

25

Draw the tree of function calls for the case of N=100 and show in each node: left, right, m, right-left+1



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
one	320	141	2	513	433	145	235
ten	2	513	320	433	235	141	145
hurend	2	141	145	235	320	433	513

$T(n,k,d) = \theta(d(n+k))$

Stable: count sort is stable → yes

Space: $\theta(n+k)$

P5. (10 points) **Bucket sort**

a) (5 pts) Assume you want to use bucket sort to sort an array A, that has integers in the range [-100, 350). (i.e. A[i]≥-100 and A[i]<350, for all valid i). You will use 50 buckets. Write the formula to find the index, `bucketIdx`, for the bucket where A[i] should go. Make sure you indicate any rounding (up or down) if necessary.

arr [ ] = {100, ..., 349) because [-100,350)

min-value = -100 | i = index of element in arr [ ]

max-value = 349 | bucket-range = ceil((max-min+1)/bucket-num)

bucket-num = 50 | or $\left\lceil \frac{max - min + 1}{bucket\text{-}num} \right\rceil$

bucketIdx = (A [I] + 100) . A.size() / max_value + 101);

= (A[I]+100).50/450.

b) (5 pts) What does the following C code print?

```
float bucket1 = (39/100)*10;
printf("\nbucket1 = %.2f\n", bucket1);
int bucket2 = (39/(float)100)*10;
printf("bucket2 = %d\n", bucket2);
int bucket3 = (float)39/100;
printf("bucket3 = %d\n", bucket3);
int bucket4 = 39/(float)100;
printf("bucket4 = %d\n", bucket4);
```

bucket1 = 0.00

bucket2 = 3

bucket3 = 0

bucket3 = 0

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, Android Platform and GNU

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

(1pts) What two sorting algorithms does it combine? Insertion sort and Merge 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? special case for 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. (6 pts) Fill in the arrays to show the required processing with count sort for the data below.

	0	1 ✓	2 ✓	3 ✓	4 ✓	5 ✓	6 ✓
Original array	C, Alice	B, Jane	A, Jane	F, John	A, Matt	D, Sam	B, Tom

Counts array after part 1 (counts of each key):

Index:	A	B	C	D	E	F
Counts array:	2	2	1	1	0	1

1st

Counts array after part 2 (after cumulative sum):

Index:	A	B	C	D	E	F
Counts array:	2	4	5	6	6	7

2nd

Show the counts array and the copy array after each of the next 2 big steps of count sort as shown in slide 6 (i.e. after a first element is placed in the copy array, and after a second element is placed in the copy array). Create columns as needed in the tables below.

Index:	A	B	C	D	E	F
Counts array:	2	③	5	6	6	7
Counts array:	2	3	5	⑤	6	7

Index:	0	1	2	3	4	5	6
Copy array:				B Tom			
Copy array:						D Sam	

Remember to include your name at the top.

Write your answers in this document or a new document called **2320_H5.pdf**. Place **median3_quick_sort.c** and **2320_H5.pdf** in a folder called **2320_HW5**, zip that and send it.

1/