1. (15%) Below is the pseudo code for Quicksort that we talked about in class. As usual with recursive functions on arrays, we see the array indices p and r as arguments. Quicksort(a, p, r) sorts the part of the array between p and r inclusively. The initial call (that is, to sort the entire array) is Quicksort(A, 0, n - 1).

```
QUICKSORT($A, p, r$)
IF $p < r$ THEN
    q=PARTITION($A, p, r$)
    QUICKSORT($A, p, q-1$)
    QUICKSORT($A, q+1, r$)
FI
```

```
PARTITION($A, p, r$)
$x = A[r]$
$i = p - 1$
FOR $j = p$ TO $r - 1$ DO
    IF $A[j] \leq x$ THEN
        $i = i + 1$
        Exchange $A[i]$ and $A[j]$
    FI
OD
Exchange $A[i+1]$ and $A[r]$
RETURN $i + 1$
```

a). Let A = {*3, 6, 1, 5, 8, 2, 4, 7, 3*}, and assume we call Quicksort(A, 0, 8). Show what happens during the first invocation of Partition. What is the value of q returned, and what are the two recursive calls made?

Answer:

p = 0, r = 8, pivot = A[r] = 3, i= -1, j = 0

Initial step:

```
 {3,  6,  1,  5,  8,  2,  4,  7,  3}
i j
```

For loop:

```
 {3,  6,  1,  5,  8,  2,  4,  7,  3}
  i   j
 {3,  6,  1,  5,  8,  2,  4,  7,  3}
  i       j
 {3,  1,  6,  5,  8,  2,  4,  7,  3}
      i       j
 {3,  1,  6,  5,  8,  2,  4,  7,  3}
      i           j
 {3,  1,  6,  5,  8,  2,  4,  7,  3}
      i               j
 {3,  1,  2,  5,  8,  6,  4,  7,  3}
          i               j
 {3,  1,  2,  5,  8,  6,  4,  7,  3}
          i                   j
```

p = 0, r = 8, pivot = A[r] = 3, i= -1, j = 0

Initial step:

```
    {3,  6,  1,  5,  8,  2,  4,  7,  3}
 i   j
```

For loop:

```
    {3,  6,  1,  5,  8,  2,  4,  7,  3}
     i   j
    {3,  6,  1,  5,  8,  2,  4,  7,  3}
     i       j
    {3,  1,  6,  5,  8,  2,  4,  7,  3}
         i       j
    {3,  1,  6,  5,  8,  2,  4,  7,  3}
         i           j
    {3,  1,  6,  5,  8,  2,  4,  7,  3}
         i               j
    {3,  1,  2,  5,  8,  6,  4,  7,  3}
             i               j
    {3,  1,  2,  5,  8,  6,  4,  7,  3}
             i                   j
    {3,  1,  2,  5,  8,  6,  4,  7,  3}
             i                       j
    {3,  1,  2,  3,  8,  6,  4,  7,  5}
             i                       j
```

a). (6%) Let A = {4, 7, 2, 6, 9, 3, 5, 2, 4}, and assume we call Quicksort(A, 0, 8). Show what happens during the first invocation of Partition. What is the value of q returned, and what are the two recursive calls made?

Answer:

Loop terminates, then exchange A[i+1] and A[r]:

```
    {3,  1,  2,  3,  8,  6,  4,  7,  5}
             i                       j
```

The above is the first invocation of Partition returning new pivot position at:
 q = i+1 = 3

The next two recursive calls are:

QUICKSORT(A, p, q-1) = QUICKSORT(A, 0, 2)
QUICKSORT(A, q+1, r) = QUICKSORT(A, 4, 8)

b). (4%) What is the running time of Quicksort when all elements of array A have the same value?

Answer:

- QUICKSORT will have the worst case running time $O(n^2)$ when all elements of array A have the same value.
- All elements compare equal values to the pivot for every partition resulting in i will go through the whole array to the position r-1, and then q = r will always return as the pivot position.
- This causes unbalanced partition that the right subarray after partition will have length of 1, and the left subarray's length always have n-1.

b). (4%) What is the running time of Quicksort when all elements of array A have the same value?

Answer:

Here is the analysis of $O(n^2)$ for it:

Assume that there are n elements in array A,

```
QUICKSORT(A, p, r)                  ----> T(n)
IF p <r THEN
        q = PARTITION(A, p, r)      ----> O(n)
        QUICKSORT(A, p, q-1)        ----> T(n-1)
        QUICKSORT(A, q+1, r)        ----> T(0)
```

We get $T(n) = O(n) + T(n-1) + T(0) = O(n^2)$

Recursive partition on the left subarray will have the same worst partitioning:

$n + (n-1) + (n-2) + ... + 1 = n(n+1)/2 \implies O(n^2)$

c). (4%) Briefly sketch why the running time of Quicksort is Θ(n^2) when the array A contains distinct elements and is sorted in decreasing order.

Answer:

- Compare with the previous subquestion, the performance of running time for QUICKSORT with all elements in decreasing order is still O(n^2).
- Assume A = [6, 5, 4, 3, 2, 1] n = 6
- After the initial loop A_1 = [1], A_{12} = [6, 5, 4, 3, 2, 7] n = 6
- After the 2nd loop A_{21} = [1], A_{22} = [6, 5, 4, 3, 2] , A_{23}=[7] n = 5
- After the 3rd loop A_{31} = [1], A_{32} = [2], A_{33}= [5, 4, 3, 2, 6] , A_{33}=[7] n = 4
- :::
- :::
- etc

c). Briefly sketch why the running time of Quicksort is Θ(n^2) when the array A contains distinct elements and is sorted in decreasing order.

Answer:

Compare with the previous subquestion, the performance of running time for QUICKSORT with all elements in decreasing order is still O(n^2).

Assume A = [6, 5, 4, 3, 2, 1] n = 6

After the initial loop A_1 = [1], A_{12} = [6, 5, 4, 3, 2, 7] n = 6

After the 2nd loop A_{21} = [1], A_{22} = [6, 5, 4, 3, 2] , A_{23}=[7] n = 5

After the 3rd loop A_{31} = [1], A_{32} = [2], A_{33}= [5, 4, 3, 2, 6] , A_{33}=[7] n = 4

:::

etc

This will lead to unbalanced partitions:

```
QUICKSORT(A, p, r)                 ----> T(n)
IF p <r THEN
        q = PARTITION(A, p, r)       ----> O(n)
        QUICKSORT(A, p, q-1)         ----> T(0)
        QUICKSORT(A, q+1, r)         ----> T(n-1)
```

We get T(n) = O(n) + T(0) + T(n-1) = O(n^2)

2. (5%) Here is an array which has just been partitioned by the first step of Quicksort:

2, 0, 1, 3, 4, 7, 6, 5, 8

Which of these elements could have been the pivot? (if there are more than one possibility, list them all)

Answer:

3, 4, 8

Explanation: everything to the left of x must be < x, and everything to the right of x must be > x.

3: A[2, 0, 1, 8, 4, 7, 6, 5, 3]

4: A[2, 0, 1, 3, 8, 7, 6, 5, 4]

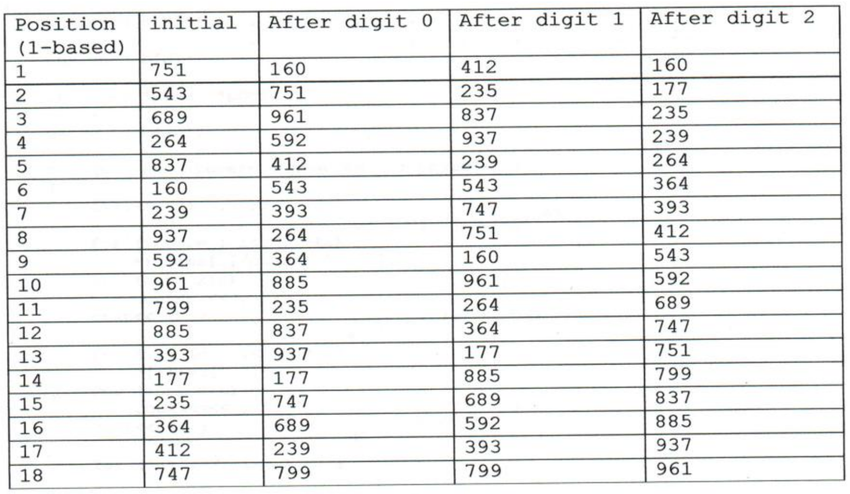
5: A[2, 0, 1, 3, 4, 7, 6, 5, 8]

H1.9

3. (10%) Sort the following array using radix sort with base 10. For each digit, show the array C used to count and order after sorting each digit.

{751, 543, 689, 264, 837, 160, 239, 937, 592, 961, 799, 885, 393, 177, 235, 364, 412,747 }

Answer:



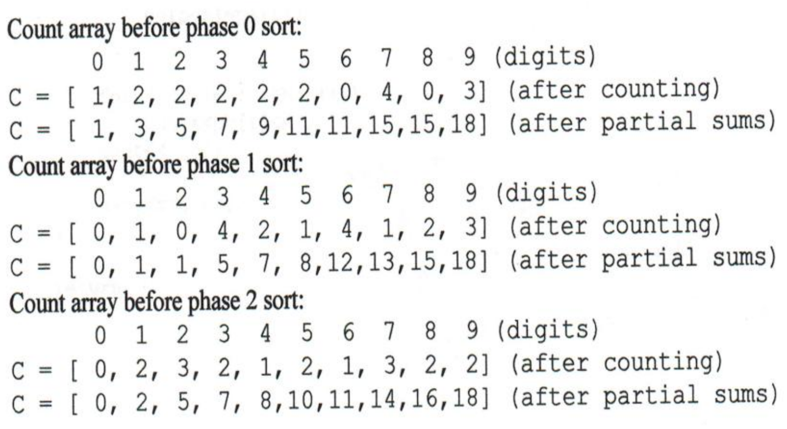
Position (1-based)	initial	After digit 0	After digit 1	After digit 2
1	751	160	412	160
2	543	751	235	177
3	689	961	837	235
4	264	592	937	239
5	837	412	239	264
6	160	543	543	364
7	239	393	747	393
8	937	264	751	412
9	592	364	160	543
10	961	885	961	592
11	799	235	264	689
12	885	837	364	747
13	393	937	177	751
14	177	177	885	799
15	235	747	689	837
16	364	689	592	885
17	412	239	393	937
18	747	799	799	961

H1.10

3. (10%) Sort the following array using radix sort with base 10. For each digit, show the array C used to count and order after sorting each digit.

{751, 543, 689, 264, 837, 160, 239, 937, 592, 961, 799, 885, 393, 177, 235, 364, 412,747 }

Answer:



Count array before phase 0 sort:

```
        0  1  2  3  4  5  6  7  8  9 (digits)
C = [ 1, 2, 2, 2, 2, 2, 0, 4, 0, 3] (after counting)
C = [ 1, 3, 5, 7, 9,11,11,15,15,18] (after partial sums)
```

Count array before phase 1 sort:

```
        0  1  2  3  4  5  6  7  8  9 (digits)
C = [ 0, 1, 0, 4, 2, 1, 4, 1, 2, 3] (after counting)
C = [ 0, 1, 1, 5, 7, 8,12,13,15,18] (after partial sums)
```

Count array before phase 2 sort:

```
        0  1  2  3  4  5  6  7  8  9 (digits)
C = [ 0, 2, 3, 2, 1, 2, 1, 3, 2, 2] (after counting)
C = [ 0, 2, 5, 7, 8,10,11,14,16,18] (after partial sums)
```

H1.11

4. (10%) Illustrate and show how to sort the array A = [6,8,3,1,7,4,4,5,3,2,7,1,8,4,3] using Bucket sort .

Answer: must show pseudo codes or show how it works

```
BucketSort(A)
1.  n ← A.length
2.  Let B[0..n – 1] be a new array
3.  for i ← 0 to n – 1
4.        make B[i] an empty list
5.  for i ← 0 to n
6.        do insert A[i] into list B[ ⌊nA[i]⌋]
7.  for i ← 0 to n – 1
8.        do sort list B[i] with insertion sort
9.  concatenate the lists B[i]s together in order
10. return the concatenated lists
```

Bucket sort divides the interval [0,1) into n equal-sized subintervals, or buckets, and then distributes the **n** input numbers into the buckets.

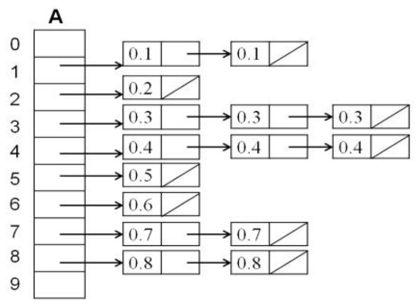
H1.12

4. (10%) Illustrate and show how to sort the array A = [6,8,3,1,7,4,4,5,3,2,7,1,8,4,3] using Bucket sort .

Answer:

Since Bucket Sort divides the interval [0,1) before sorting, we need to process A array so that each element's value fall into [0,1).
(Consider if A = [6,82,3,1123,7,4,4,5,3,2,7,1,8,4,332]?)

- A/10 = [0.6,0.8,0.3,0.1,0.7,0.4,0.4,0.5,0.3,0.2,0.7,0.1,0.8,0.4,0.3] all < 9, we can pick n = 10



- Concatenate A together in order = A [0.1, 0.1, 0.2, 0.3, 0.3, 0.3, 0.4, 0.4, 0.4, 0.5, 0.6, 0.7, 0.7, 0.8, 0.8]
- Convert A*10 = [1, 1, 2, 3, 3, 3, 4, 4, 4, 5, 6, 7, 7, 8, 8]

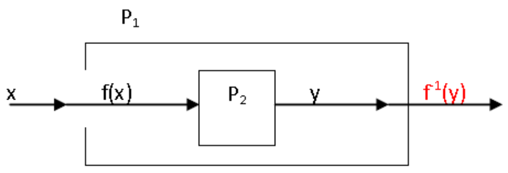
H1.13

5. (10%) Given an array of n different English words and all in lower case with different length. Suggest an algorithm and approach for sorting the words in alphabetic order in O(n) time.

Answer:

Reduction:

- Assume you have two problems P_1 and P_2 and the algorithm for solving P_2 is known.
- A ***reduction*** from P_1 to P_2 is a way to solve problem P_1 by using the algorithm for solving P_2.



- In our question the reduction is from alphabetic-sort to radix-sort:

H1.14

5. (10%) Given an array of n different English words and all in lower case with different length. Suggest an algorithm and approach for sorting the words in alphabetic order in O(n) time.

Answer:

- **Translate Input of P1 to input for P2:**
  - Let ***m*** be the maximal length of word in the input, ***m*** is constant.
  - Represent each letter as a digit in base 27, i.e., in range [0..26], where a=1, b=2,…, z =26 $\rightarrow$ O(nm) = O(n)
  - Words with k < m letters will have succeeding zeros added - O(nm)=O(n)
- **Solve P2 on the transformed input:**
  - Execute radix sort where d = *m* and b=27 $\rightarrow$ O(d(n+b))=O(n)
- **Translate output of P2 to output of P1:**
  - Change the numbers sequences back to words $\rightarrow$ O(nm)=O(n)
-
- Total run time is O(n).
-

5. (10%) Given an array of n different English words and all in lower case with different length. Suggest an algorithm and approach for sorting the words in alphabetic order in O(n) time.

Answer:

```
findLongestWord(){
var EnglishStr = [“i”, “like”, “professor”, “wu”, “who”, “teach”, “algorithms”,
    “because”, “handsome”, “professional”,...];

var longestWord = 0;

for(var i = 0; i < EnglishStr.length; i++){

if(EnglishStr[i].length > longestWord){ // EnglishStr[i].length is greater than
    the word it is compared with...

longestWord = EnglishStr[i].length; // ...then longestWord takes this new value
}
}   //Total run time is O(n).
```

5. (10%) Given an array of n different English words and all in lower case with different length. Suggest an algorithm and approach for sorting the words in alphabetic order in O(n) time.

Answer:

Example:

Alphabetic-sort input: {blue, red, green}

Radix-sort input: { (2,12, 21,5,0), (18,5,4,0,0), (7,18,5,5,14) }

Radix-sort output: { (2,12, 21,5,0), (7,18,5,5,14) , (18,5,4,0,0) }

Alphabetic-sort output: {blue, green, red}

6.(10%)Suppose you have an array of a million English vocabulary Strings to sort, but you happen to know that there are only four different varieties of Strings in it: "Family", "Hero", "Monkey", and "Together". You want to sort this vocabulary array to put all the "Family" words first, then all the "Hero", then all the "Monkey", then all the "Together" words. How would you do it? You could use merge sort or Quicksort and get a runtime proportional to N log N, where N is ~one million. Can you do better? Adjust your answer by illustrating how to sort it.

Answer:

- Since you have already known that there are only four different varieties of Strings in it: "Family", "Hero", "Monkey", and "Together".
- Using Bucket sort to place all Family words into F (ascii 70) slots, all Hero words into H (72) slot, and son on.

7.(10%) Show how to sort n integer numbers in the range 0 to (n^8 -1) in O(n) time?

Answer:

Comparison-based algorithm takes O(nlogn) .

Counting-sort: k = n^8 → O(n+ n^8) = O(n^8)

Radix-sort: b = n, d = 8, → d(b+n) = O(8 (n+n)) = O(n)

Why is that ?

To use radix-sort, we need to convert each integer base n, then radix sort them.

8 digits are necessary to represent n^8 -1in the radix-n, that is

n-1 n-1 n-1 n-1 n-1 n-1 n-1 n-1

Convert all numbers to base n in O(n) total time using mod and div operations.

x = [x_7,x_6,x_5,x_4,x_3,x_2,x_1,x_0] (x_0= x mod n, x_i = $\lfloor x/n^i \rfloor \bmod n$)

Call radix-sort on the transformed numbers. O(n)

All the numbers are in range 1 to n^7, therefore, we have at most 8 digits for each number. The running time for the suggest algorithm:

d = 8, b =n → O(8(n+n)) = O(n).

8. (30%) *8. (30%) Using the array A[5, 5, 0, 6, 2, 0, 1, 3, 2, 6, 1, 4, 2, 4], convert the pseudo code of Counting Sort algorithm listed in the text book to Java code and print each intermediate result to show the array C and array B step by step. (To receive full points, you must include all screen shots of Java codes and each intermediate result of array C and array B.)*

Answer: