**Assignment - 5**

1. You are given three sorted arrays of integers (in ascending order). Write a program to find a triplet (one element from each array) such that distance is minimum.

If a[i], b[j], and c[k] are three elements, then distance is calculated as triangular distance.

Distance = diff(a[i]-b[j]) + diff(b[j]-c[k]) + diff(c[k]-a[i])

Solve it using O(n) complexity algorithm.

1. Write a program to count and display the number of lines, words, and characters in a text file (similar to Unix *wc* program)
2. Write a recursive function print\_int(int n) to print a number as a character string. The number could be either a positive or negative integer. Don’t use any variable or array in the function, other than int n given as argument).
3. Use the idea from the above recursive program to write recursive function itoa( int n) to convert an integer into string.
4. Write a stable version of quicksort that requires O(n) extra space.
5. Write an in-place version of quicksort that uses only O(log n) space (except input, obviously).
6. Find the running time T for quicksort for different values of N (100000, 200000, 400000, 500000, 700000, 1000000, 1500000, 2000000, etc…). Is the ratio T/(Nlog N) approaching any constant?
7. Find the running time T for mergesort for different values of N (100000, 200000, 400000, 500000, 700000, 1000000, 1500000, 2000000, etc…). Is the ratio T/(Nlog N) approaching any constant?
8. Write a program to sort an array of positive integers using Radix sort by processing the digits using [Least Significant Digit (LSD)](http://www.computerhope.com/jargon/l/leastsd.htm) method.
9. You have 100000 positive numbers which are already sorted, find the time taken for each of quicksort, merge sort, bubble sort, counting sort, radix sort. Which is the best approach in this case? Repeat the above for 200000, 300000, and 500000 sorted numbers.
10. Write a recursive program(method given below) to rotate a string of length n left by i positions. For example, with n=7, and i = 3, string “abcdefg” after rotation becomes “defgabc”.

**Use the following method:**

Assume the array of characters is AB where A is a string of first i characters and B the

remaining part of the string. So, the problem is to rotate AB so that it becomes BA. The method is: Reverse A to get ArB; then reverse B to get ArBr; now reverse the complete string ArBr to get BA (final answer).

To implement it:

Write a recursive function **reverse(int from, int to)** that reverses the string from position *from* to *to*. For example, if the string is “abcdefg” reverse(0, 2) makes the string become “cbadefg”. Rotating the entire string by i involves the following three calls (example in comments is for string “abcdefg” and i = 3):

reverse(0, i-1) /\* cbadefg \*/

reverse(i, n-1) /\* cbagfed \*/

reverse(0, n-1) /\* defgabc, final answer \*/

1. You have n distinct integers, each in the range [0, 9,999,999]. (n<=10000000). You have to write a time and space efficient variation of counting sort (assuming the memory is scarce). Instead of using an index in the integer array C to denote if the corresponding number is present in the input, you have to use a single bit for each input integer. So each integer in array C (assuming int size is 32 bits) of 32 bits can be used to represent presence/absence of 32 input numbers. For example, if the input contains the number 2, then the third least significant bit from right (LSB) of the first integer will be set to 1. (All the bits of array are initially cleared). Similarly, if the input contains number 32, then the most LSB of second integer is set. Each cell in the figure below denote a bit:

+----+----+-------+----+----+----+

C[0]: | 31 | 30 | . . . | 02 | 01 | 00 |

+----+----+-------+----+----+----+

C[1]: | 63 | 62 | . . . | 34 | 33 | 32 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ….. |  |  |  |  |  |

………… so on………………..

Take input n as command-line argument and generate n distinct random numbers.

After all the input numbers are scanned and the corresponding bits are set in C, you can just scan through C, and print the position of the bits that are set to 1 (this is similar to counting sort, but using bits instead of indices of array C).

You need to use bit-setting operations.

For setting a bit of C corresponding to input number i, use

void set\_bit(int i)

{

C[i>>5] |= (1 << (i & 0x1f)); /\* bitwise SHIFT, OR, AND operators \*/

}

For initializing the bit vector C, each bit must be set to 0:

void clear\_bit(int i)

{

C[i>>5] &= ~(1<<(i & 0x1f)); /\* bitwise SHIFT, AND, NEGATION operators \*

}

For testing if a bit is set

int test\_bit(int i)

{

return C[i>>5] & (1<<(i & 0x1f));

}

NOTE: Right shifting a positive number by 5 is equivalent to dividing by 32 and discarding the remainder. This gives the index of C that we are concerned with.

AND-ing i with 0x1f (11111) will keep only the rightmost 5 bits and discard the remaining bits ( rightmost 5 bits as they 25 is 32). These rightmost five bits of i will give the position of the bit in the index of C that needs to be set/clear/tested in array C.

The pseudo-code of the above method looks like:

Given array C[10000000]

generate n random numbers

for all i in 0 …9999999

clear\_bit(i) /\* clear bit of array C \*/

for each j in n

set\_bit(j) /\* set bit of array C \*/

/\* print sorted numbers \*/

for all i in 0 …9999999

if test\_bit(i) EQUALS 1 /\* test bit of array C \*/

print i

Now you need to generate n distinct random positive integers in the range 0…n (be careful here), and apply the above method to sort them.

1. Implement the normal count sort, and compare its running time with that of the above program for n = 100000, 200000, 500000, 900000.
2. Compare the running time of the above variation of count sort (problem 12) with that of quicksort for n = 100000, 200000, 500000, 900000.
3. Find the running time T for the above variation of count sort (problem 12) for different values of N (100000, 200000, 400000, 500000, 700000, 1000000, 1500000, 2000000, etc…).. Is the ratio T/N approaching any constant?
4. Write a program to reverse individual words in a given sentence.

**Input/Output:**

Enter a sentence: How are you

Reversed Individual Words: you are How

1. You are given a dictionary file containing words, each in separate lines. Assume each word is less than 50 letters and each letter is among a, b, c,…, z. So, “apple” upon sorting becomes “aelpp”. Many words in the dictionary will result in the same string after sorting. For example, sorted string of both “cat” and “act” will be “act”. Similarly, sorted string of both “listen” and “silent” is “eilnst”. These actual words are anagrams whose sorted string is same. You have to write a program to read each word from a file wordfile.txt (provided to you), sort the word, and print the sorted word and actual word Use Unix pipes for redirecting outputs. So the print would actually be used to send output to next program (look at the command at the end). So, write two programs, one for reading files and sorting each word and printing sorted string and the actual word in a line.

printf(“%s %s\n”, sortedword, word);)

1. Call this file sortwords.c. Compile and create executable from this file called sortwords.

The above program’s output can be fed to *sort* program of Unix. (see the command at the end).

(b)Now, write another program called anagram.c (its executable called print\_anagrams) that will get the input from the *sort* program (*sort* program sorts all the lines of input line by line, so each anagram will become contiguous after *sort*) The output from the *sort* program is taken by program (called) anagrams, and it should print all the anagrams in a separate line. Save the output of anagrams to anagrams.txt.

The command will look like:

./sortwords < wordfile.txt | sort | ./print\_anagrams > anagrams.txt

An example follows:

wordfile.txt

act

apple

boy

cat

stop

stray

tops

output of sortwords that will be sent to *sort*

act act

apple aelpp

boy boy

act cat

opst stop

arsty stray

opst tops

Each line above is a pair of *sorted-string actual-word*

The *sort* sorts the lines and the result will be:

act act

act cat

aelpp apple

arsty stray

boy boy

opst stop

opst tops

The above would then be used by ./anagrams to print all anagrams in a single line. The output of ./anagrams would be sent to file anagrams.txt which should now contain:

act cat

apple

stray

boy

stop tops

See that each line contains actual words that are anagrams.

The wordfile.txt will be uploaded to CMS.