CS 213 : Data Structures & Algorithms Minor

Practice Problems Sheet 1 [January 22, 2020]

Problem 1 : 1, 4, 9,.... are numbers that are known as **perfect squares**. We would like to know the density of perfect squares in the universe of natural numbers. Specifically, it is required to list all perfect squares (including how many) among natural numbers with d decimal digits, 5 <= d <= 10.

Problem 2 : The number 6, has an interesting property that sum of all its factors (except the number itself) is the given number (for ex, 6 = 1 + 2 + 3). Numbers that satisfy this property are known as **perfect numbers**. Another perfect number is 28. It is desired to find all perfect numbers that are computationally possible in 8 byte unsigned integers.

Problem 3 : The 3-digit number 153 has the following interesting property: $1^3 + 5^3 + 3^3 = 153$. In other words, the sum of the digits, raised to the power of the number of digits, is the number itself. One is curious to know whether this is an exception or there are other numbers with the same characteristic. In particular, find and report all such numbers with d decimal digits, 3 <= d <= 10.

Problem 4 : Determine all prime numbers, having d decimal digits. Compare experimentally the density of **prime numbers** with that of **perfect squares** in the universe of natural numbers for a given d.

Problem 5 : A pair of prime numbers are called **twin primes** if they differ by 2, for example (5,7), (11, 13) are twin primes. Find and report all twin primes having d decimal digits.

Problem 6 : A reputed mathematician claims that **reciprocals of the divisors of a perfect number must add up to 2**. She illustrates this for the first two perfect numbers :

- For 6, we get 1 / 6 + 1 / 3 + 1 / 2 + 1 / 1 = 2;
- For 28, we get 1 / 28 + 1 / 14 + 1 / 7 + 1 / 4 + 1 / 2 + 1 / 1 = 2

One wishes to verify experimentally whether her claim is true for other perfect numbers also.

Problem 7 : **Mersenne number**, named in honour of Father Marin Mersenne (1588 - 1648), is a positive integer that is one less than a power of two. A **Mersenne prime** is a Mersenne number that is a prime, it is of the form $2^p - 1$, where p is a prime.

The first four Mersenne primes are $3(2^2 - 1)$, $7(2^3 - 1)$, $31(2^5 - 1)$ and $8191(2^{13} - 1)$. However, $2047(2^{11} - 1)$ is composite having 23 and 89 as factors. Only 47 such primes are known till October 2009, the last one discovered is $(2^{43,112,609} - 1)$ which is also the largest known prime known till date.

All the Mersenne primes in the last decade have been discovered by GIMPS (Great Internet Mersenne Prime Search) a distributed computing project on the internet. To get a feel for the size of the 47^{th} Mersenne prime, it has more than 10 million decimal digits (needs 3461 pages with 75 decimal digits per line and 50 lines per page for its representation).

Find and report as many Mersenne primes as you possibly can in 8 byte unsigned integer. For every integer of the form $2^p - 1$, where p is a prime, display its prime factors if it is not a Mersenne prime.

Problem 8: **Amicable numbers** are two different numbers that are related such that the sum of the proper divisors (proper divisor of a number is a positive integer divisor other than the number itself, e.g. the proper divisors of 6 are 1, 2, and 3.) of each is equal to the other. This concept is related to that of a perfect number, which is a number which equals the sum of *its own* proper divisors (said to form an aliquot sequence of period 1), amicable numbers form an aliquot sequence of period 2.

For example, (220, 284) is the smallest pair of amicable numbers . Sum of proper divisors of 220 is 284 = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110; while sum of proper divisors of 284 is 220 = 1 + 2 + 4 + 71 + 142. Around 12,000,000 amicable pairs were known till 2007. The first few amicable pairs are: (220, 284), (1184, 1210), (2620, 2924), (5020, 5564), (6232, 6368).

Determine and report as many amicable numbers as you possibly can.

Problem 9. Sociable numbers are generalizations of the concepts of amicable numbers and perfect numbers. A set of sociable numbers is a kind of aliquot sequence, or a sequence of numbers each of whose numbers is the sum of the factors of the preceding number, excluding the preceding number itself. For the sequence to be sociable, the sequence must be cyclic, eventually returning to its starting point.

The period of the sequence, or order of the set of sociable numbers, is the number of numbers in this cycle.

If the period of the sequence is 1, the number is a **sociable number of order 1**, or a perfect number —for example, the proper divisors of 6 are 1, 2, and 3, whose sum is again 6. A pair of **amicable** numbers is a set of **sociable numbers of order 2**. There are no known sociable numbers of order 3.

It is an open question whether all numbers end up at either a sociable number or at a prime (and hence 1). Or equivalently, whether there exists a number whose aliquot sequence never terminates.

An example with period 4:

The sum of the proper divisors of 1264460 ($2^2 * 5 * 17 * 3719$) is:
1 + 2 + 4 + 5 + 10 + 17 + 20 + 34 + 68 + 85 + 170 + 340 + 3719 + 7438 + 14876 + 18595 + 37190 + 63223 + 74380 + 126446 + 252892 + 316115 + 632230 = 1547860

The sum of the proper divisors of 1547860 (2^2 * 5 * 193 * 401) is:

1 + 2 + 4 + 5 + 10 + 20 + 193 + 386 + 401 + 772 + 802 + 965 + 1604 + 1930 + 2005 + 3860 + 4010 + 8020 + 77393 + 154786 + 309572 + 386965 + 773930 = 1727636

The sum of the proper divisors of 1727636 (2^2 * 521 * 829) is:

1 + 2 + 4 + 521 + 829 + 1042 + 1658 + 2084 + 3316 + 431909 + 863818 = 1305184

The sum of the proper divisors of 1305184 (2^5 * 40787) is:

1 + 2 + 4 + 8 + 16 + 32 + 40787 + 81574 + 163148 + 326296 + 652592 = 1264460.

Determine and report as many scoiable numbers of order ≥2 as you possibly can.

P10. Given a number as input find and report the prime exponent representation of the number. For the input 180, the expected output is of the form

Prime exponent representation of 180 is

prime 2 exponent 2

prime 3 exponent 2

prime 5 exponent 1

180 has three prime factors and sum of exponents is 5

P11. Write following functions for working with strings

- **bool isnum(string s)** : given a string s, this function will examine all the characters in it and return true if all symbols are decimal digits [0..9] and false otherwise. For example, for the string "1234.56" it returns false whereas for "1230" it returns true. Assume that sign is not permitted in the parameter s
- **bool isreal(string s)** : given a string s, this function will examine all the characters in it and return true if the symbols determine a real number and false otherwise. Note that the symbol (.) must be present in the string and must be either preceded or followed by decimal digits. For example, for the string "1234.56" it returns true whereas for "1230" it returns false. Assume that sign is not permitted and exponent form of floating number is also not allowed in the parameter s.
- **string trimall(string s**) : The purpose of this function is to trim leading and trailing whitespaces in the parameter s. For example, if the string is " \nabcd \n" then trimall() will return the string "abcd" and for " 123 45 ", it will return "123 45". It removes only the leading and trailing whitespaces and not those inside.
- Write a main_program that tests all the above functions with set of appropriate values and do the necessary corrections, if required, to get the desired effect.

P12. Write following functions for working with strings

- **int stoi(string s)** : given a string s, this function (a) checks that isreal() is true, and (b) when isreal(s) is true, constructs and returns the floating point number corresponding to s else it returns -1.0. For example, for the string ".1234" it returns the float value 0.1234, for the string "002030.", it returns the float value 2030.0 and for "1234", it returns -1.0. Assume that sign is not permitted in the parameter s. The purpose of this function is to convert a string to its equivalent integer number.
- **flost stof(string s)** : given a string s, this function (a) checks that isnum() is true, and (b) when isnum(s) is true, constructs and returns the integer number corresponding to s else it returns -1. For example, for the string "1234" it returns the integer 1234, for the string "002030", it returns the integer 2030 and for "123 4", it returns -1. Assume that sign is not permitted in the parameter s. The purpose of this function is to convert a string to its equivalent floating number.
- Write a main_program that tests both the above functions with a set of distinct values and do the necessary corrections, if required, to get the desired effect.

P13. The poem, **Palanquin Bearers,** by Sarojini naidu, is given below in an unformated text form.

Palanquin Bearers
Sarojini Naidu

Lightly, O lightly we bear her along, She sways like a flower in the wind of our song; She skims like a bird on the foam of a stream, She floats like a laugh from the lips of a dream. Gaily, O gaily we glide and we sing, We bear her along like a pearl on a string.

Softly, O softly we bear her along, She hangs like a star in the dew of our song; She springs like a beam on the brow of the tide, She falls like a tear from the eyes of a bride. Lightly, O lightly we glide and we sing, We bear her along like a pearl on a string.

The usual formatted pretty form is

Palanquin Bearers

Lightly, O lightly we bear her along,
She sways like a flower in the wind of our song;
She skims like a bird on the foam of a stream,
She floats like a laugh from the lips of a dream.
Gaily, O gaily we glide and we sing,
We bear her along like a pearl on a string.

Softly, O softly we bear her along,
She hangs like a star in the dew of our song;
She springs like a beam on the brow of the tide,
She falls like a tear from the eyes of a bride.
Lightly, O lightly we glide and we sing,
We bear her along like a pearl on a string.

Sarojini Naidu

Processing of text and formatted output as per the instructions given is the programming problem here. There are 4 popular compositions from well known poets, given in the files, bachchan.txt, patriotism.txt, sarojini.txt and tagore.txt. The guidelines for formatting the poems are

(a) Every line begins with a word whose first letter is in uppercase; but words with leading uppercase letter may also be present in the middle of a line,

(b) A line may end with the punctuation symbols {comma(,) semicolon(;) fullstop(.) exclaimation (!) question mark(?) colon(:) single-quote (') double-quote (“) } or a space

(c) A blank line between texts indicates a new paragraph.

(d) The minimum number of words in a line, ignoring the punctuation symbols in it, in the entire poem, is given in the following table.

Poem	**bachchan.txt**	**patriotism.txt**	**sarojini.txt**	**tagore.txt**
Min words in a line for the poem	**3**	**4**	**7**	**10**

Download the files, from the directory, four_poems, from the course page. This problem involves reading and processing a few text files, each of which contains a poem. Write a program that given a text file as input (a) reads it line by line and rearranges it in the form of a poem using the guidelines provided, (b) display through the error channel (so that the formatted output is not corrupted) for each paragragh and for each formatted line the number of words it it, not counting the symbols, and also the maximum and minimum words in its lines to be displayed at the end of the paragraph. For example, at the end of first paragraph of Palanquin Bearers poem, the following displays through the output and error channel are desired.

OUTPUT CHANNEL	ERROR CHANNEL
Lightly, O lightly we bear her along,	Para 1 : Line 1 : words : 7
She sways like a flower in the wind of our song;	Para 1 : Line 2 : words : 11
She skims like a bird on the foam of a stream,	Para 1 : Line 3 : words : 11
She floats like a laugh from the lips of a dream.	Para 1 : Line 4 : words : 11
Gaily, O gaily we glide and we sing,	Para 1 : Line 5 : words : 8
We bear her along like a pearl on a string.	Para 1 : Line 6 : words : 10
	Number of Lines : 6
	Max words : 11 Min words : 7

The formatted version of the poems are given in the directory named “formatted” for your reference.

P14. A file containing a few function definitions are given followed by the main() is given to you. Assume that the program given is free from all errors and that functions are defined before they are called. You are given a function name as input. Write a program that does the following processing :

(a) Searches for the definition of the function name given as input. If the definition is found in the file, it reports in the format, else reports that there is no definition.

Function name ..... defined in lines ..... through .....

(b) examines the rest of the file to identify calls to this function and reports in the format :

Function name ..... is called ..... number of times.

line .... : call is ..... in function .....

line .... : call is .....in function .....

.....

Function name ..... called ..... number of times in function .....

The above reporting is done for all the functions where the input function is called.

Generate one or more files with function definitions and calls, run your program and store input and output in the same file.

P15. The received call numbers log of a mobile phone is made available for processing. The following format is to be used for each entry of the log.

(a) Phone_number (b) Date (c) Time (d) Duration (e) Name

Each entry is stored in a line and there are 1 or more blanks separating the fields. Date is stored in the form dd/mm/yyyy and Time is in the format hh/mm am/pm, Duration is in seconds.

Write a program that when given

(a) a Phone_number as input, displays (i) all the entries in the log for this number, (ii) count of the number of calls and (iii) the entry with the longest duration

(b) a Date as an input, displays (i) all the entries in the log for this date, (ii) count of the number of calls and (iii) the entry with the longest duration.

Generate one or more files with log entries as specified above, run your program and store input and output in the same file.

P16. A log of emails is given to you for processing. The format of the email log is given to be of the following form. The first line has 8 fields each separated from the next with one or more spaces, i) From ii) email_address iii) Day iv) Month, v) Date, vi) Time in hh:mm:ss, vii) year , viii) ignore this field. The second line as shown below has a few fields but no new information that is not captured in line 1. The third line gives the details of the recipientand the fourth line the subject. While this is not the exact structure of email logs, it is adequate for our processing requirements. There may be any number of empty lines within an email message.

From xy@yz.uvw.pqr Sun Aug 5 21:20:41 2018 +0530

Date: Sun, 5 Aug 2018 21:20:41 +0530 (IST)

To: Supratim Biswas <sb@cse.iitb.ac.in>

Subject: Happy Friendship Day

.....

Write a program that reads in a log file of emails in the format specified above and displays the following information when given (i) an email address as input and (ii) a date as input

(a) All emails from this address with day, date, time and subject, one line per mail

(b) Number of mails sent on the date given.

Generate one or more files with log entries as specified above, run your program and store input and output in the same file.

P17. Consider the problem of writing a function that when given a polynomial and its degree, constructs and returns the derivative of the polynomial and its degree.

```
#include <simplecpp>
const int SIZE = 10;
void derivpoly (float p[],  int deg, float dp[], int & dervdeg)
{ // Given a polynomial p of degree deg constuct the
   // derivative polynomial, dp, of degree dervdeg = deg -1
  for (int i = 0; i < deg; i++)  dp[i] = (i+1)*p[i+1];
  dervdeg = deg -1 ;
  return;
}
```

Examine the signature of the definition of function derivpoly() given above. There are 4 parameters of this function and the parameter passing mechanisms for each are :

- float p[] indicates that the function accepts a float array passed by reference
- int deg indicates that the second parameter passed by value
- float dp[] indicates that the 3rd parameter is also a float array passed by reference
- int & dervdeg indicates that the 4th argument is also passed by reference

Note that through the third parameter the function accepts an integer as passed by value from the caller. The changes made by the function to this parameter would have no effect on the corresponding actual parameter value of the caller. The call by value may be interpreted as a one-way communication between the caller to the callee.

The remaining parameters are all passed by reference, the syntax used in 1st and 2nd parameters are used for passing arrays using reference. The 4th parameter, int &, where & stands for a special type called a reference type in C++, shows the syntax to be used for passing a scalar by reference. The implication of using a '&' after the type or not is the difference between pass by reference and by value. A parameter passed by reference from a caller to its callee, results in the changes made by the callee to this object, visible to the caller, when the control returns back to the caller.

Run the main() given below with sample values to check the correctness of your function definition and get familiar with writing functions with one or more return values as you need. This skill is required in many problems in this sheet.

```
int main()
{
   float coef[SIZE], deriv_coeff[SIZE];
   int degree, derivdeg;
   cout << " give degree of poly : <= 9 : ";
   cin >> degree; // Compare degree with SIZE and do the needful
```

```
    for (int i = 0; i <= degree; i++) cin >> coef[i];
    derivpoly(coef, degree, deriv_coeff, derivdeg);
    cout << " Degree of given polynomal : " << degree
        << " With coefficients :  “ << endl;
    for (int i = 0; i <= degree; i++) cout <<  coef[i] << " " << endl;
    cout << " Degree of  derivative polynomal : " << derivdeg
        << " With coefficients :  " << endl;
    for (int i = 0; i <= derivdeg; i++) cout <<  deriv_coeff[i] << " " << endl;
    return 0;
}
```

Run the main_program with sample values to check the correctness of the function and get familiar with writing functions with one or more return values as you need. This skill is required in most of the problems in this sheet.

P18. *(a) Given an integer array write three separate functions to find the number of i) strictly positive values (> 0) ii) zero values and iii) strictly negative values (< 0)in the array. Function main should call the three functions and display the required numbers.*

For example, if the inputs are 10 (number of elements in the array) and the elements

-5 25 0 -5 0 0 21 6 25 -5

The expected output of the three functions in the program are :

Number of strictly positive values : 4

Number of zero values : 3

Number of strictly negative values : 3

(b) Now change the definition of all the three functions so that they return in addition to the count in part (a) above, an array that holds the indices where the desired type of values are found.

For the input of part (a), the expected output is :

Number of strictly positive values : 4

The strictly positive values occur at : 1, 6, 7, 8

Number of zero values : 3

The zero values occur at : 2, 4, 5

Number of strictly negative values : 3

The strictly negative values occur at : 0, 3, 9

P19. Examine the sequential search program given below.

```
// include files omitted
const int SIZE=100;
bool seqsearch(int [], int , int ); // prototype given, definition later
int main()
{ int array[SIZE];
  int num, value, i;
  cout << " Give number of elements in array : ";
  cin >> num;
  cout << " Key in the array elements ";
  for (i= 0; i < num ; i++ ) cin >> array[i]; cout << endl ;
  cout << " Give the value to be searched : " << endl;
  cin >> value; cout << endl ;
  if ( seqsearch(array, num, value)) // call function and check return value
        cout << value << " found in array " << endl;
  else    cout << value << " not found in array " << endl;
  return 0;
}

// definition of seqsearch() follows

bool seqsearch(int b[], int last, int elm)
{ int i;
  bool found;
  found = false; // default - value is absent
  for ( i = 0; i < last ; i++ )
  { if ( b[i] == elm )
      { found = true; return found; }
  };
  return found; // value not found
}
```

(a) Compile and run the program for a few sample inputs and make sure that it works properly for both successful and unsuccessful search. Make necessary changes if required.

(b) Rewrite the function seqsearch() so that it returns a single integer value denoting the number of times the element (to be searched) is found in the array, 0 denoting absence.

(c) Change function seqsearch() so that it returns the index of the first occurence of the element (being searched) in addition to the result returned by it – for example return value is true indicating present and the function also gives back a number, say 5, indicating that the first position the element being searched occurs in the array, is at index 5.

(d) Similar to (c) above, except that if the element being searched is found in the array, then the function communicate to its caller using an array that has all the positions where this element is found.

P20. *Given an input, scalar or an array, write functions to construct and return an array with the property specified against each. Also write a main() to test out your function.*

(a) Given an integer, the function returns an array with the digits stored in different locations of the array and also the number of digits in the number. For example if the input is 2343012, the function should return an array with a[0] = 2, a[1]=1, a[2]=0, ..., a[6]=2, and digits = 7. Essentially the array is to be populated by extracting the digits from right to left.

(b) Given an integer as input, the function returns an array with the factors stored in different locations of the array, in ascending order of factors, and also the total number of factors of the number.

(c) Given a number n, write a function that returns an array of all prime numbers with n digits and also returns the number of such primes found.

(d) Given 2 arrays, say a[] and b[] as input which represent polynomials, write separate functions for finding the sum, difference and product over polynomials.

(e) Given a string and a delimiter, write a function that returns an array of substrings of the given string separated using the delimiter. It also returns the number of such substrings found. For example, given the path as

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/profbiswas/ipe/cs101

*the function should return an array, say path, such that **/usr/local/sbin** is in path[0], **/usr/local/bin** in path[1], **/usr/sbin** in path[2], **/usr/bin** in path[3], **/sbin** in path[4], **/bin** in path[5] and **/home/profbiswas/ipe/cs101** in path[6] and that there are 7 such substrings.*

P21. Change functions seqsearch() and main() given in **P19** such that searching over arrays of data types, other than integers, are possible. Show the program for data types from among the following set of types { unsigned char, signed long, double, string}. Use template function.

P22. Horner's scheme for evaluating a polynomial, with real coefficients, at a given point (a real), is supposed to be implemented in the following function.

```
float polyeval(float b[], int n, float val) // b[0] to b[n] are coefficients of the polynomial p(x)
{ float result;                              // compute p(val) and return by Horner's scheme
  result = b[n];
  for (int i = n; i>=1; i--)  { result = result*val + b[i-1]; }
  return result;
```

}

(a) Check the function for correctness, making changes as appropriate, write a main_program that calls it by passing a polynomial and a point, and displays the return value of the function.

(b) Write another function that computes the value of a polynomial p(x) at x = a by the explicit formulation, i. e., p(a) = b[0] + b[1]*a + b[2]*a*a + ... + b[n]*a *a ....*a. There should be no attempt to reuse and reduce the computations involved in the implementations of this function. Write a main_program to test the working of your program.

(c) Run both the functions separately over polynomials ranging from degree 1 to degree 5 by generating inputs and time each run. Examine the results and write a summary of your findings.

P23. *Consider the program given in the notes that is supposed to find a root using the method interval halving. You have to make a better product for this technique for finding all roots of a function. You may use the following guideline in this respect.*

- *Write a function that given the n+ 1 real coefficients of a n degree polynomial, using first 2 parameters, two points a and b through the 3rd and 4th parameter, uses interval halving to return a root to the accuracy specified as the fifth parameter. A value 2 for this parameter implies that accuracy of the root detected is given as correct up to 2 places of decimal or equivalently the difference between two successive approximations to the root is less than 10 $^{-2}$*
- *Write a function that given the n+ 1 real coefficients of a n degree polynomial, first 2 parameters, finds and returns two reals, a and b, such that p(a)*p(b) < 0. Even gross values of a and b are acceptable here so long they trap an odd number of roots.*
- *Write a function that given a polynomial and its degree as its parameters, say p(x) of degree n and also a real root of the polynomial, say r, constructs and returns a new polynomial, say q(x), such that the root r has been removed from p(x). Essentially, q(x) = p(x) / (x-r)*
- *Write another function that given an array of n roots of a polynomial, constructs and returns the coefficients of the n degree polynomial with the same roots as given in the input.*
- *Write a main() that calls the functions listed above including others that you may have used in your design to (a) read a n degree polynomial, extract a root at a time and report all the n roots, (b) uses the roots found in part (a) above, to reconstruct a polynomial with these roots, compares the coefficients and reports the differences found.*

P24. *Graephe's root squaring method for simultaneous detection of all roots of a polynomial is required. Read the notes on moodle for details.*

(a) Write down the equations of the form in mathematical form for a given polynomial, exactly as in the illustration given in the notes : $p(x)=\sum_{i=0}^{i=n} a_i x^i$

Rearranging the odd and even terms, we get,

$$(a_1 x + a_3 x^3 + a_5 x^5 + ...)^2 - (-a_0 - a_2 x^2 - a_4 x^4 - ....)^2 = \sum_{i=0}^{i=2n} b_i x^i$$

Equating the coeffcients from both sides, gives a set of equations :

$$b_0 = -a_0^2 ; b_1 = a_1^2 - 2a_0 a_2 ; b_2 = -a_2^2 + 2a_1 a_3 - 2a_0 a_4 ;$$

$$b_3 = a_3^2 - 2a_2 a_4 + 2a_1 a_5 - 2a_0 a_6 ; .....$$

T*he equations reveal a pattern for computing the coefficients b_k of the polynomial after one step of the root squaring process has been applied. Each coefficient b_k has a square term a_k^2 and several products terms $a_p * a_q$where the sunscripts p and q of a have a relation to the subscript k of b. Make obsevations to generalize and formulate a general relation.*

(b) Write a function that given a polynomial and its degree, applies one step of the root squaring process and returns the coefficients of the new polynomial of the same degree.

(c) Write a function that given a polynomial and its degree, returns an array that contains approximations of all the roots of the input polynomial, as explained in the notes.

(d) Write a main_program that reads a polynomial and its degree and a specified accuracy and uses the functions written above to find approximations to all roots of the given polynomial.

P25. *Given an array of integers, sort the array in ascending or descending order as asked using Bubble sort. A function named as bubble(), given below, is available for examination and reuse if you wish to.*

```
void bubble (int a[], int size)// array a[0] to a[n-1]
  {   int pass, j, tmp;
      for (pass = 1; pass < size ; pass++)
      {
         for (j=0; j < size; j++)
         {  if (a[j] > a[j+1])
                { tmp = a[j]; a[j] = a[j+1]; a[j+1] = tmp;  } /* end of if */
         } /* end of inner for loop */
      } /* end of outer for loop */
  } /* end of bubble sort */
```

(a). Check the order, ascending or descending, in which the given function is sorting its argumenmt array a[]. Write a main_program to call bubble() and display the array beore and after sorting.

(b) The given function can do with some improvements (i) it does not use the fact that the size of each pass reduces by 1, and (ii) that if there is not a single interchange during a certain pass, then the array is already sorted and we can return without completing the remaining passes. Test the performance with a main_program.

(c) Take the efficient version of bubble sort designed and tested in part (b) above and write another function for sorting in the reverse order.

(d) Is it possible to combine the two functions of parts (b) and (c) into a single function that does the sorting in ascending or descending order as desired.

(e) Rewrite the efficient bubble sort function so that it may be applied to at least three data types from among the following set of types { unsigned char, signed long, double, string}.

P26. *Radix sort algorithm was discussed in the notes. It is desired to implement this sorting algorithm over integral data types. A design involving a few functions is outlined below. Using this scheme or a different approach of your own, write a program that radix sorts its input.*

- A function that given a number and a position returns the digit at that position in the number
- A function that given an array and its size, i) distributes the elements into ten buckets based on the digit at a given place (unit, tens, hundreds, ...) and ii) reconstructs an array by merging the buckets in order from 0 to 9.
- a main_program that does the sorting by using the above functions

P27. Write recursive functions for computing the following

i) n! , factorial of n, n>=0

ii) gcd (m, n) ;, where m and n are non-negative integers

For each of the recursive functions written by you, write a main program to perform the following computations :

i) find $\binom{n}{r}$, where n and r are both non-negative

ii) find the gcd and lcm of n numbers given as input

iii) find $\sum_{k \le n} \binom{r+k}{k}$ and verify whether it is same as $\binom{r+n+1}{r}$ for non-negative integers r and n

P28. Write recursive functions for the following recursive formulations :

(i) $F_n = F_{n-1} + F_{n-2}$; n > 1; $F_0 = 0$; $F_1 = 1$

(ii) $G_n = G_{n-1} + n - 1$; n >=2; $G_1 = 0$

(iii) $H_n = 2*H_{n-1} + 1$; n > 1; $H_1 = 1$

(iv) Ackermann function; A(m, n) defined for all m, n >= 0 as follows

A(m, n) = n+1; m = 0

A(m, n) = A(m-1, 1); m > 0 and n = 0

A(m, n) = A(m-1, A(m, n-1)); m > 0 and n > 0

Write main programs that display all the above sequences upto a given integer n.

P29(a). Write a recursive function for searching an element in a sorted array of size n using binary search. Write a main program that (i) reads an array of a basic type, and an element of the same type as that of array elements, (ii) sorts the array using any sorting algorithm of your choice, and (iii) finds the presence or absence of the element in sorted array using binary search.

(b) Generalize the design of part (a) above so that it works for all types for all the three subparts mentioned (Hint define and use template functions).

(c) Modify the recursive binary search function so that the position in the array where the element is found is also reported.

P30. You are given a file, student.txt, which has 4 words per line, roll, name, surname and attendance. You are expected to use only public data members and a single function member, the constructor function. All functions that you may need are to be defined as non-member functions.

(a) Create a new data type, say, **student,** that has exactly four public members and a public constructor function; you may overload the constructor if you so wish. Declare an array of type student, max size is 100; e. g., **student cs213m[100];**

(b) Read the data from the file student.txt and initialize the array with data as provided in the file; assume that the input file has no errors. Traverse the array to display the values stored in the array. You may directly use cin with cs213m[i].roll, cs213m[i[.name, cs213m[i].surname, .. etc. (no need to overload the operators << and >>)

(c) Use index sorting method to create an index array that would display the records in i) ascending order of roll number, ii) ascending order of name, iii) ascending order of suurname, or iv) descending order of attendance.

(d) Write a main program and show that your design works as per specifications.

Input file : student.txt

160001000 Ashish Tiwari 35

160220005 Ankita Sharma 34

160220010 Dhruv Dhar 31

150020003 Moses Apurva 25

Sample code for index sort is given for your reference.

```
class record {
public :
  int roll; // 7 digit
  string name; // concatenation of all parts of the name
  int total;
};

void index_sort(int a[], record rec[],int n)
{int pass, j;
 int tmp;
 bool xchange = true;
 for (pass = 1; pass < n && xchange == true; pass++)
  { xchange = false;
    for (j=0; j < n - pass; j++)
     { if (rec[a[j]].name > rec[a[j+1]].name)
        { tmp = a[j]; a[j] = a[j+1]; a[j+1] = tmp;  xchange = true;
        } /* end of if */
     } /* end of inner for loop */
  } /* end of outer for loop */
} /* end of bubble sort */

int main()
{
  record data[20];
  int i = 0;
  while ( cin >> data[i].roll >> data[i].name >> data[i].total)
  {// display each record
    cout << " roll = " <<  data[i].roll  << " name = " <<  data[i].name
          << " total = " << data[i].total << endl;
     i ++;
  };
  // index sort on total
  int count = i;
  cout << " no of records = " << count << endl;
```

```
int ind1[20];
for ( int i = 0 ; i < count ; i++ ) ind1[i] = i;
index_sort(ind1, data, count );

// display the sorted data by index sorting
cout << endl << " sorted output " << endl << endl;
cout <<  " Roll no.  " << " Name "  << "  Total " << endl ;
for ( int i = 0; i < count ; i++ )
   cout <<  data[ind1[i]].roll  <<  data[ind1[i]].name  << data[ind1[i]].total << endl;
}
```

P31. We wish to create a new data type, **points,** which is essentially a pair of integers; the purpose is to abstract the 2-dimensional euclidean space. You are expected to use only public data members and a single function member, the constructor function. All functions that you may need are to be defined as non-member functions.

(a) Declare an array set of type points, max size is 100; e. g., points **set[100];**

(b) Read a given number of points (pairs of integers), from the file points.txt and initialize the array with data as provided in the file; assume that the input file has no errors. Traverse the array to display the points stored in the array. You may directly use cin or cout to access set[i].x, set[i].y(no need to overload the operators << and >>)

(c) Use index sorting method to create an index array that would display the records in i) ascending order of x-values, ii) ascending order of y-values. Also add the following non-member functions for processing a set of points, i) the point with the smallest x-value, ii) the point with the smallest y-value, iii) given a x-value, the number of points with this value, iv) given a y-value, the number of points with this y-value, and v) given two points (x1, y1) and (x2, y2), the euclidean distance between them, $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$

(d) Write a main program that after reading and initializing an array of points, displays the points in (i) sorted on x-values, (ii) sorted on y-values, and (iii)finds the two points with the largest distance detween them.

P32. It is required to create a data type for polynomials in a single variable x. The public data members are float array of coefficients and the degree as an integer. A public constructor should create a zero polynomial of degree 0 as default. Design and implement a class for polynomials that permits the following processing on this data type.

(a) reading and displaying a polynomial

(b) functions that perform polynomial arithmetic - addition, multiplication and division

(c) a main program that tests the class design.

P33. Consider an array of values of a certain data type where a value may be present more than once. Modify the program of P34 (b) and (c) to work for such inputs and return the subarray (all consecutive indices) where the element occurs in the array. The program is expected to be applied to arrays of various basic types.

P34. Overload the operators << and >> in the program of P35 written by you and make it work for your class.

******** END OF DOCUMENT *********