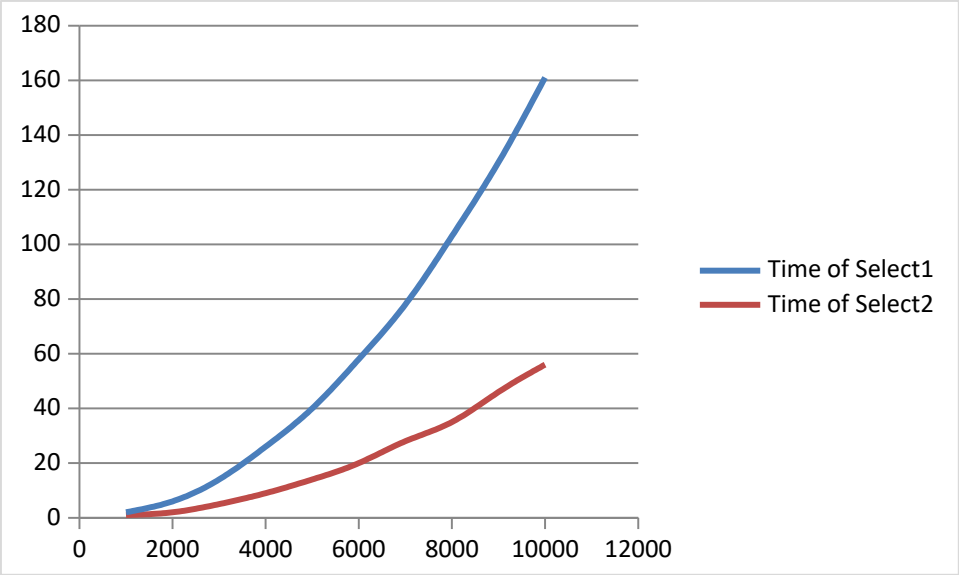
CHAPTER 1

Introduction

1.1

```
/*
   Exercise 1.1
   Selection of integers with k = N/2
     select1 => sorting and selecting
        select2 => keeping top k
*/

#include <iostream>
#include <ctime>
#include <cmath>
#include <vector>
#include <algorithm>
using namespace std;



void sort(vector<int> & vec)
{ // bubble sort ascending
  bool sorted = false;
  while (!sorted)
  {
       sorted = true;
       for (auto i = 1; i < vec.size(); i++)
       {
         if (vec[i-1]> vec[i])
         {
        swap(vec[i],vec[i-1]);
             sorted = false;
         }
       }
  }
}

void sortDec(vector<int> & vec)
{ // bubble sort descending
  bool sorted = false;
  while (!sorted)
  {
       sorted = true;
       for (auto i = 1; i < vec.size(); i++)
       {
         if (vec[i-1]< vec[i])
         {
        swap(vec[i],vec[i-1]);
             sorted = false;
         }
       }
  }
}
```

```
 int  select1(vector<int>  nums)
{
  int k = (nums.size()+1)/2;
  sort(nums);
  return nums[k];
}


  int  select2(const vector<int> &nums)
{
       int k = nums.size()/2;
       vector<int> topK(nums.begin(), nums.begin() + k);

       sortDec(topK);
       for (auto i = k; i < nums.size(); i++)
       {
              if (nums[i] > topK[k-1])
              {
                     for (auto j = k-2; j >=0 ; j--)
                            if (nums[i] < topK[j])
                            {topK[j+1] = nums[i]; break;}
                            else
                                   topK[j+1] = topK[j];
                     if (topK[0] < nums[i])
                            topK[0] = nums[i];
              }
       }
       return topK[k-1];
}

int main()
{
  vector<int> nums;
  int selected;
  time_t start, end;

  srand(time(NULL));
  for (auto numInts = 1000; numInts<=10000; numInts+=1000)
         // sizes 1,000, 2,000, 3,000, ...10,000
  {
       nums.resize(numInts);

     start = time(NULL);
     for (auto i = 0; i < 10; i++) // run 10 times
        {
         for (auto j = 0; j < numInts; j++)
                nums[j] = rand()%(2*numInts);
         selected = select1(nums); // or selected = select2(nums);
        }
     end = time(NULL);
        cout<<numInts<<"\t"<<difftime(end,start)<<endl;
  }
  return 0;
}
```



2.
```
/*
  Word Puzzle problem
  from the example in figure 1.1
*/

#include<iostream>
#include<fstream>
#include<string>
#include<vector>
#include "matrix.h"
#include<algorithm>

using namespace std;

const int MAXROWS = 4;
const int MAXCOLS = 4;

struct Orientation
{
      Orientation() : delRow(0), delCol(0) {}
Orientation operator() (int direction)
  {
    switch (direction)
   {
       case 0 : delRow = -1; delCol = -1; break;
       case 1 : delRow = -1; delCol =  0; break;
       case 2 : delRow = -1; delCol =  1; break;
       case 3 : delRow = 0; delCol =  -1; break;
       case 4 : delRow = 0; delCol =   1; break;
       case 5 : delRow = 1; delCol =  -1; break;
       case 6 : delRow = 1; delCol =   0; break;
       case 7 : delRow = 1; delCol =   1; break;
   }
       return *this;
  }
  int delRow;
  int delCol;
```

```
};

class Puzzle
{
public:
    Puzzle(int numRows, int numCols )
       {
              matrix<char> temp(numRows,numCols);
              puzzle= temp;
              initPuzzle();
       }
       Puzzle(int numRows , int numCols , vector<string> wordList) : dictionary(wordList)
     {
          matrix<char> temp(numRows,numCols);
              puzzle= temp;
              initPuzzle();
        }
       void solvePuzzle();
       void findWords(int startRow, int startCol, Orientation orient);
private:
   void initPuzzle();
   matrix<char> puzzle;
   vector<string> dictionary;
};

void Puzzle::initPuzzle()
{
  puzzle[0][0] = 't';
  puzzle[0][1] = 'h';
  puzzle[0][2] = 'i';
  puzzle[0][3] = 's';
  puzzle[1][0] = 'w';
  puzzle[1][1] = 'a';
  puzzle[1][2] = 't';
  puzzle[1][3] = 's';
  puzzle[2][0] = 'o';
  puzzle[2][1] = 'a';
  puzzle[2][2] = 'h';
  puzzle[2][3] = 'g';
  puzzle[3][0] = 'f';
  puzzle[3][1] = 'g';
  puzzle[3][2] = 'd';
  puzzle[3][3] = 't';
}

void Puzzle::solvePuzzle()
{
  Orientation orient;
  for ( auto startRow = 0; startRow < puzzle.numrows(); startRow++)
         for ( auto startCol=0; startCol < puzzle.numcols(); startCol++)
               for (auto i = 0; i < 8 ; i++)
                     findWords(startRow,startCol,orient(i));
}


void Puzzle::findWords(int startRow, int startCol, Orientation orient)
{
  string word ="";
```

```
 int row = startRow;
 int col = startCol;
 do
        {
               word = word + puzzle[row][col];
               if (find(dictionary.begin(), dictionary.end(), word) != dictionary.end())
                      cout<<word<<" found starting at ("<<startRow<<","<<startCol<<")\n";
               row += orient.delRow;
               col += orient.delCol;
        }  while (row > -1 && col > -1 && row < puzzle.numrows() && col <
puzzle.numcols());
}

int main()
{
 string diction[] = {"this", "two", "fat", "fats", "at", "wad", "ad", "hat", "that",
"his","is","it","ah"} ;
 vector<string> dictionary(diction,diction+ 12);
 Puzzle puzzle(MAXROWS, MAXCOLS, dictionary);

  puzzle.solvePuzzle();

  return 0;
}
```

1.3

```
void printDouble(double x)
{
  if (x < 0)
  {
         cout<<"-";
         x = -x;
  }
   int intPart = floor(x);
   double fract = x - intPart;

   printOut(intPart);
   cout<<".";
   while (fract<1 && fract > 0.0000000001)// 0.0000000001 is machine accuracy.
   {
       fract *= 10;
       printDigit(floor(fract));
       fract = fract - floor(fract);
   }
  }
```

1.4

The general way to do this is to write a procedure with heading

```
void processFile( String fileName );
```

which opens *fileName*, does whatever processing is needed, and then closes it. If a line of the form

```
#include SomeFile
```

is detected, then the call

```
processFile( SomeFile );
```

is made recursively. Self-referential includes can be detected by keeping a list of files for which a call to *processFile* has not yet terminated, and checking this list before making a new call to *processFile*.

1.5

```
int ones( int n )
{
  if( n < 2 )
    return n;
  return n % 2 + ones( n / 2 );
}
```

1.6

```
void permute(const string & str, int low, int high)
{
  char letter;
  string tmp = str;
   if (low >= high)
        cout<<str<<endl;
   else
   {
     for (auto i= low; i < str.size(); i++)
        {
      swap(tmp[0], tmp[i]);
         permute(tmp, low+1, high);
        }
   }
}


void permute(const string & str)
{
  permute(str, 0, str.size());
}
```

1.7 **(a)** The proof is by induction. The theorem is clearly true for $0 < X \leq 1$, since it is true for $X = 1$, and for $X < 1$, $\log X$ is negative. It is also easy to see that the theorem holds for $1 < X \leq 2$, since it is true for $X = 2$, and for $X < 2$, $\log X$ is at most 1. Suppose the theorem is true for $p < X \leq 2p$ (where p is a positive integer), and consider any $2p < Y \leq 4p$ ($p \geq 1$). Then $\log Y = 1 + \log(Y/2) < 1 + Y/2 < Y/2 + Y/2 \leq Y$, where the first inequality follows by the inductive hypothesis.

(b) Let $2^X = A$. Then $A^B = (2^X)^B = 2^{XB}$. Thus $\log A^B = XB$. Since $X = \log A$, the theorem is proved.

1.8 **(a)** The sum is 4/3 and follows directly from the formula.

(b) $S = \frac{1}{4} + \frac{2}{4^2} + \frac{3}{4^3} + \cdots$. $4S = 1 + \frac{2}{4} + \frac{3}{4^2} + \cdots$. Subtracting the first equation from the second gives $3S = 1 + \frac{1}{4} + \frac{2}{4^2} + \cdots$. By part (a), $3S = 4/3$ so $S = 4/9$.

(c) $S = \frac{1}{4} + \frac{4}{4^2} + \frac{9}{4^3} + \cdots$. $4S = 1 + \frac{4}{4} + \frac{9}{4^2} + \frac{16}{4^3} + \cdots$. Subtracting the first equation from the second gives $3S = 1 + \frac{3}{4} + \frac{5}{4^2} + \frac{7}{4^3} + \cdots$. Rewriting, we get $3S = 2\sum_{i=0}^{\infty} \frac{i}{4^i} + \sum_{i=0}^{\infty} \frac{1}{4^i}$. Thus $3S = 2(4/9) + 4/3 = 20/9$. Thus $S = 20/27$.

(d) Let $S_N = \sum_{i=0}^{\infty} \frac{i^N}{4^i}$. Follow the same method as in parts (a) – (c) to obtain a formula for S_N in terms of S_{N-1}, S_{N-2},..., S_0 and solve the recurrence. Solving the recurrence is very difficult.

1.9 $\sum_{i=\lfloor N/2 \rfloor}^{N} \frac{1}{i} = \sum_{i=1}^{N} \frac{1}{i} - \sum_{i=1}^{\lfloor N/2-1 \rfloor} \frac{1}{i} \approx \ln N - \ln N/2 \approx \ln 2.$

1.10 $2^4 = 16 \equiv 1 \pmod 5$. $(2^4)^{25} \equiv 1^{25} \pmod 5$. Thus $2^{100} \equiv 1 \pmod 5$.

1.11 **(a)** Proof is by induction. The statement is clearly true for $N = 1$ and $N = 2$. Assume true for $N = 1, 2, \ldots, k$. Then $\sum_{i=1}^{k+1} F_i = \sum_{i=1}^{k} F_i + F_{k+1}$. By the induction hypothesis, the value of the sum on the right is $F_{k+2} - 2 + F_{k+1}$ $= F_{k+3} - 2$, where the latter equality follows from the definition of the Fibonacci numbers. This proves the claim for $N = k + 1$, and hence for all N.

(b) As in the text, the proof is by induction. Observe that $\phi + 1 = \phi^2$. This implies that $\phi^{-1} + \phi^{-2} = 1$. For $N = 1$ and $N = 2$, the statement is true. Assume the claim is true for $N = 1, 2, \ldots, k$.

$$F_{k+1} = F_k + F_{k-1}$$

by the definition, and we can use the inductive hypothesis on the right-hand side, obtaining

$$\begin{aligned} F_{k+1} &< \phi^k + \phi^{k-1} \\ &< \phi^{-1}\phi^{k+1} + \phi^{-2}\phi^{k+1} \\ F_{k+1} &< (\phi^{-1} + \phi^{-2})\phi^{k+1} < \phi^{k+1} \end{aligned}$$

and proving the theorem.

(c) See any of the advanced math references at the end of the chapter. The derivation involves the use of generating functions.

1.12 **(a)**
$$\sum_{i=1}^{N}(2i-1) = 2\sum_{i=1}^{N} i - \sum_{i=1}^{N} 1 = N(N+1) - N = N^2.$$

(b) The easiest way to prove this is by induction. The case $N = 1$ is trivial. Otherwise,

$$\begin{aligned}
\sum_{i=1}^{N+1} i^3 &= (N+1)^3 + \sum_{i=1}^{N} i^3 \\
&= (N+1)^3 + \frac{N^2(N+1)^2}{4} \\
&= (N+1)^2\left[\frac{N^2}{4} + (N+1)\right] \\
&= (N+1)^2\left[\frac{N^2+4N+4}{4}\right] \\
&= \frac{(N+1)^2(N+2)^2}{2^2} \\
&= \left[\frac{(N+1)(N+2)}{2}\right]^2 \\
&= \left[\sum_{i=1}^{N+1} i\right]^2
\end{aligned}$$

1.15

```
class EmployeeLastNameCompare
{
  public:
  bool operator () (const Employee & lhs, const Employee & rhs) const
    { return getLast(lhs.getName())< getLast(rhs.getName());}
};

string getLast( const string & name)
{
  string last;
  int blankPosition = name.find(" ");
  last = name.substr(blankPosition+1, name.size());
  return last;
}
int main()
{
  vector<Employee> v(3);
  v[0].setValue("George Bush", 400000.00);
  v[1].setValue("Bill Gates", 2000000000.00);
  v[2].setValue("Dr. Phil", 13000000.00);
  cout<<findMax(v, EmployeeLastNameCompare())<<endl;
  return 0;
}
```

1.16

```
matrix() : array(10)
```

```
{for( auto & thisRow : array )
thisRow.resize( 10 );
}

void resize(int rows, int cols)
{
  array.resize(rows);
  for ( auto & thisRow : array)
         thisRow.resize(cols);
}
```