BASIC ALGORITHMS

Dr. Jozo Dujmović

Operations with arrays:

1. Read / Initialize / Write
2. Reduction (reduce to scalar)
3. Selection (select specific elements)
4. Compression (remove selected elements)
5. Expansion (insert additional elements)
6. Reverse (the first element becomes last)
7. Rotate left/right (circular shift)
8. Sort (increasing or decreasing)
9. Merge (combine two arrays)
10. Search

Read an array

```
void ReadArray(int a[], int& na)
{
  int i;
  cout << "\n\nNumber of components of array = ";
  cin >> na;
  cout << "\nArray = ";
  for(i=0; i<na; i++) cin >> a[i];
}
```

Random initialization of an array

```
#include <stdlib.h> // or <ctime>

void RandomArray(int a[ ], int na)
{
   int i;
   randomize( ); // or srand(time(NULL))
   for(i=0; i<na; i++)
     a[i] = rand()%100;  // 0<=a[i]<=99
}
```

Display an array

```
void show(char s[], int x[], int nx)
{ int i;
  cout << "\n" << s << "[0.." << nx-1 << "]=";
  for(i=0; i<nx; i++) cout << ' ' << x[i] ;
  cout << endl;
}

void DisplayArray(int a[], int na)
{  int i;
   cout << "Array[0.."<< na-1 << "] = ";
   for(i=0; i<na; i++)
     cout << a[i] << (((i+1)%15) ? ' ' : '\n');
   cout << "\n\n";
}
```

The concept of accumulator:

Accumulator is a variable that accumulates temporary results of a sequence of operations with components of an array and at the end contains a final result

```
ACC = <neutral element for
        selected operator>;

for(i=0; i<na; i++)
   ACC = ACC <operator> a[i];

cout <<"\nResult = " << ACC;
```

Array reduction: sum

```
sum=0;
for(i=0; i<na; i++)
    sum += a[i];
cout <<"\nSum = " << sum;
```

Array reduction: product

```
prod=1.;
for(i=0; i<na; i++)
    prod *= a[i];
cout <<"\nProd= " << prod;
```

Array reduction: maximum

```
max=a[0];
for(i=1; i<na; i++)
    if(a[i]>max) max=a[i];
cout <<"\nMax = " << max;
```

Array reduction: minimum

```
min=a[0];
for(i=1; i<na; i++)
    if(a[i]<min) min=a[i];
cout <<"\nMin = " << min;
```

Array reduction: max and min

```
min=max=a[0];
for(i=1; i<na; i++)
   if(a[i]<min)
      min=a[i];
   else
      if(a[i]>max)max=a[i];
```

Array selection

// Make array b from selected (e.g. odd) elements of array a

```
nb=0;
for(i=0; i<na; i++)
   if(a[i]%2) b[nb++]=a[i];
show("b", b, nb);
```

Array compression

```
// Omit components that don't satisfy some conditions
// (e.g. omit all negative and zero components)

j=0;
for(i=0; i<na; i++)
    if(a[i]>0) a[j++] = a[i];
na = j;  // new size of the
         // compressed array a[]
show("a", a, na);
```

Array expansion

```
// E.g. insert an element e in a sorted
// array b[ ] so that the array remains
// sorted
e = 3;                  // select any value
i =  nb-1;              // last element
while(i>=0 && b[i]>e)
   { b[i+1] = b[i]; i--; }  // move
b[i+1] = e;             // insert the element e
nb++;                   // update the size
show("b", b, nb);
```

Array reverse

```
// Swap first and last component, etc.

for(i=0, j=nb-1; i<j; i++, j--)
    {t=b[i]; b[i]=b[j]; b[j]=t;}

show("b", b, nb);
```

Array rotate left (one position)

```
// Cyclic rotation (one position left)

t=b[0];
for(i=0; i<nb-1; i++) b[i]=b[i+1];
b[nb-1]=t;

show("b",b,nb);
```

Array rotate right (one position)

```
// Cyclic rotation (right)

t=b[nb-1];
for(i=nb-1; i>0; i--)
   b[i]=b[i-1];
b[0]=t;

show("b",b,nb);
```

Selection sort of an array : $O(n^2)$

```
show("a",a,na);

for(i=0; i<na-1; i++)
  for(j=i+1; j<na; j++)
    if(a[j]<a[i])
      {t=a[i]; a[i]=a[j]; a[j]=t;}

show("a",a,na);
```

Swap

```
void swap(int& x, int& y)
{
  int t=x; x=y; y=t;
}

void swapint(int& x, int& y)
{
  // Swap for integers without using t:
  x=x+y; y=x-y; x=x-y;
}
```

(swap is general and swapint works only for integers)

Bubble sort of an array: $O(n^2)$

```
void BubbleSort(int a[], int na)
{ int i,
       done = 0;     // Sort termination flag
  while((! done) && (na > 1))
  {
    done = 1;
    for(i=0; i < na-1; i++)
       if(a[i] > a[i+1])
       {
          swap(a[i], a[i+1]); done = 0;
       }  // Now a[na-1] = max(a[0 .. na-1])
    na--; // Reduce size and repeat
  }
}
```

Testing whether an array is sorted

```
int test(int a[ ], int na)
{ int i,
  for(i=0; i < na-1; i++)
     if(a[i] > a[i+1]) return 0;
  return 1;  // It is sorted
}
```

Merging sorted arrays

```
void merge (int a[ ], int na, int b[ ], int nb, int c[ ], int& nc)
{ int  i,j ;              // Run time proportional to nc ( = na+nb )
  nc = i = j = 0 ;
  while (i<na && j<nb) c[nc++] = (a[i]<b[j]) ? a[i++] : b[j++];
  while (i<na)              c[nc++] = a[i++] ;      // copy
  while (j<nb)              c[nc++] = b[j++] ;      // copy
}
```

Linear search of an unsorted array : O(n)

```
// Looking for specific element E (in an
// unsorted array a[ ])

E = 3;   // Element being searched

for(i=0; i<na; i++)
   if(a[i] == E) break;

if(i==na)
   cout <<"\nElement not found\n";
else
   cout << "\na[" <<i << "] = " << E << endl;
```

Binary search of a sorted array : $O(\log_2(n))$

```
int bsearch(int v[ ], int n, int x)        // Run time is
{ int low, high, mid ;                      // proportional to
  low = 0 ; high = n-1 ;                    // log n
  while (low <= high)
  {  mid = (low + high) / 2 ;
     if      (x < v[mid])  high = mid-1 ;
     else if (x > v[mid])  low  = mid+1 ;
     else return mid ;
  }
  return -1 ;     // no match
}
// Testing the sorted array c[0 .. nc-1]
for (i=0; i<nc; i++)
      if (c[i] != c[bsearch(c, nc, c[i])])
        { cout <<"\nIterative binary search error! i = " << i ;  return 1; }
```

Recursive binary search of a sorted array: $O(\log_2(n))$

```
int bsearch(int v[ ], int low, int high, int x)
{ int mid = (low + high) / 2;
  if(low>high) return -1;          // no match
  if(x<v[mid]) return bsearch(v, low, mid-1, x);
  if(x>v[mid]) return bsearch(v, mid+1, high, x);
  return mid;
}
```

```
// Testing the sorted array c[0 .. nc-1]
for (i=0; i<nc; i++)
      if (c[i] != c[bsearch(c, 0,  nc-1, c[i])])
        { cout <<"\nRecursive binary search error! i = " << i ; return 1; }
```

Combinations: chose 3 out of n elements

$$\binom{n}{3} = \frac{n(n-1)(n-2)}{1 \cdot 2 \cdot 3}, \qquad \binom{5}{3} = \frac{5 \cdot 4 \cdot 3}{1 \cdot 2 \cdot 3} = 10$$

```
n=5
for(i=1; i<=n-2; i++)
  for(j=i+1; j<=n-1; j++)
    for(k=j+1; k<=n; k++)
      cout << i << j << k << ' ';
```

123 124 125 134 135 145
234 235 245
345

Binary variations with repetitions

```
for(i=0; i<2; i++)
  for(j=0; j<2; j++)
    for(k=0; k<2; k++)
      cout << i << j << k << '\n';
```

```
000
001
010
011
100
101
110
111
```

Permutations of 3 elements

```
for(i=1; i<=3; i++)
  for(j=1; j<=3; j++)
    for(k=1; k<=3; k++)
     if(i!=j && i!=k && j!=k)
      cout << i << j << k << '\n';
```

123
132
213
231
312
321

3! = 6 (selected from 27 combinations;
an inefficient solution)

Permutations of 4 elements

```
for(i=1; i<=4; i++)
 for(j=1; j<=4; j++) if(j!=i)
  for(k=1; k<=4; k++) if(k!=i && k!=j)
   cout <<i <<j <<k << 24/(i*j*k) <<' ';
```

```
1234 1243 1324 1342 1423 1432
2134 2143 2314 2341 2413 2431
3124 3142 3214 3241 3412 3421
4123 4132 4213 4231 4312 4321
```

4! = 24 ; This solution is more efficient
(but also slightly more complex)

Inverse digits of a positive integer (1)

```
#include<iostream>
#include<iomanip>
using namespace std;

int inverse(int n)
{       int inv=0;
        do
                inv = 10*inv + n%10;
        while(n /= 10);
        return inv;
}

int width(int n) // n>0
{       int length=0;
        do
          length++;
        while(n /= 10);
        return length;
}
```

Inverse digits of a positive integer (2)

```
int main(void)
{
  int n=1;
  while(1)
  {
     cout << "\nPositive integer n = ";
     cin >> n;
     if(n < 1) break;
     cout << n << " inverse = "
          << setw(width(n))
          << setfill('0') << inverse(n);
  }
  return 0;
}
```

Inverse digits of a positive integer (3)

```
Positive integer n = 123
123 inverse = 321
Positive integer n = 654321
654321 inverse = 123456
Positive integer n = 123000
123000 inverse = 000321
Positive integer n = 1000
1000 inverse = 0001
Positive integer n = 111
111 inverse = 111
Positive integer n = 0
Press any key to continue
```

Time measurements: seconds and wait

```
#include <iostream>
#include <ctime>  // time(), ctime(), ...
#include <string>
using namespace std;

double sec( )
{  return clock()/(double(CLOCKS_PER_SEC));
}

double msec( )
{  return 1000.*clock()/(double(CLOCKS_PER_SEC));
}

void wait(double millisec)
{  double t=msec();
   while(msec()-t < millisec);
}
```

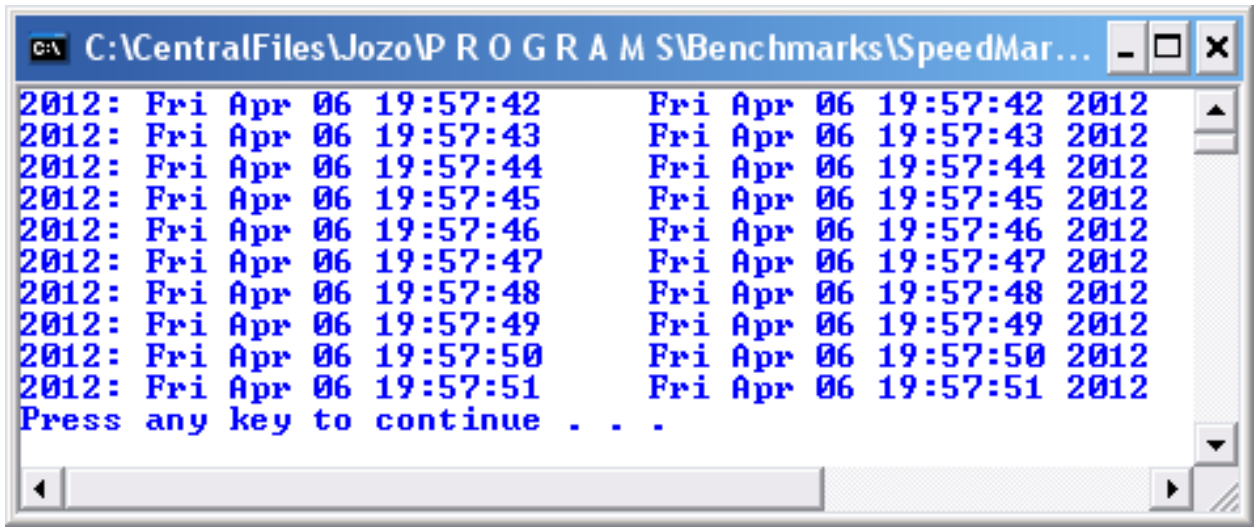
Time measurements: time stamp

```
string TimeNow(void)
{  long sec;
   time(&sec); // sec = # of seconds since 1/1/1970 0:0
   return ctime(&sec); // Includes \n at the end
}

string TimeStamp(void)
{  long sec;
   time(&sec);  // Number of seconds since 1/1/1970 0:0
   string T = ctime(&sec);
// T format: "DDD MMM dd hh:mm:ss yyyy\n\0"
   string YEAR(T, 20,4);
   string TIME(T, 11,9);
   string DATE(T, 0,11);
   return YEAR + ": " + DATE + TIME;
}
```

Time measurements: sample output

```
int main(void)
{  double t=sec();
   while(sec() - t < 10.)
   {   cout << TimeStamp() + "    " + TimeNow();
       wait(1000.);
   }
   system("pause");
   return 0;
}
```



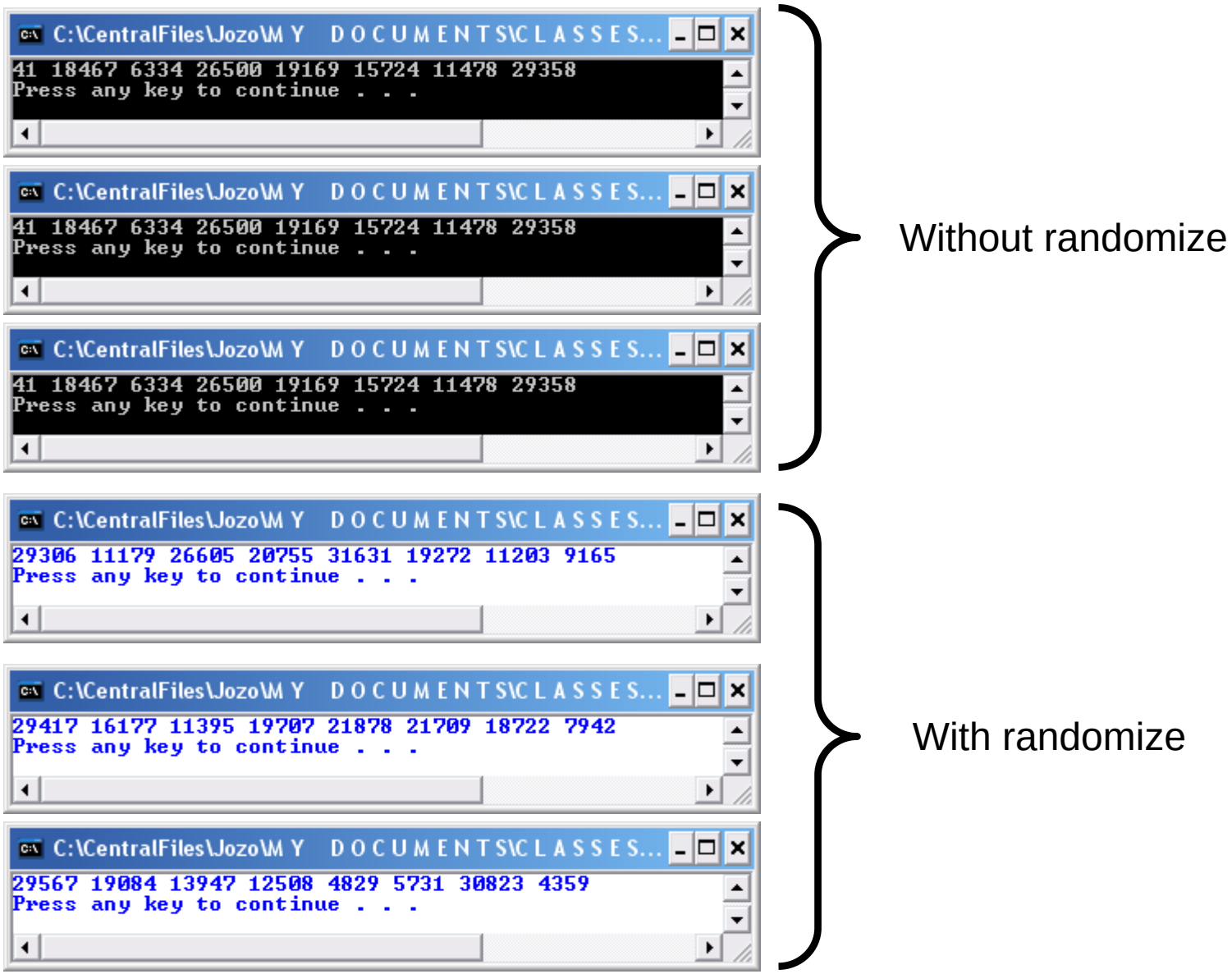
Randomize using srand(time(NULL))

```
#include <iostream>      // include standard I/O routines
#include <ctime>         // time()
using namespace std;     // allow access to all names of the std namespace

void randomize(void)     // Home-made randomize
{
    srand(time(NULL));   // Current time (seconds) used as the
}                        // seed for random number generator

int main( )
{
  randomize();           // Run this program with and without randomize
                         // to see the effect of randomizing
  for(int i=0; i<8; i++) cout << rand() << ' '; cout << endl;

  system("pause");
  return 0;              // Return the exit code 0 to the OS
}
```


C:\CentralFiles\Jozo\MY DOCUMENTS\CLASSES...
41 18467 6334 26500 19169 15724 11478 29358
Press any key to continue . . .
C:\CentralFiles\Jozo\MY DOCUMENTS\CLASSES...
41 18467 6334 26500 19169 15724 11478 29358
Press any key to continue . . .
C:\CentralFiles\Jozo\MY DOCUMENTS\CLASSES...
41 18467 6334 26500 19169 15724 11478 29358
Press any key to continue . . .
Without randomize
C:\CentralFiles\Jozo\MY DOCUMENTS\CLASSES...
29306 11179 26605 20755 31631 19272 11203 9165
Press any key to continue . . .
C:\CentralFiles\Jozo\MY DOCUMENTS\CLASSES...
29417 16177 11395 19707 21878 21709 18722 7942
Press any key to continue . . .
C:\CentralFiles\Jozo\MY DOCUMENTS\CLASSES...
29567 19084 13947 12508 4829 5731 30823 4359
Press any key to continue . . .
With randomize