Session 9
Quicksort & Searching Algorithms

Instructor: Emeterio Navarro

Date: 12/05/2012

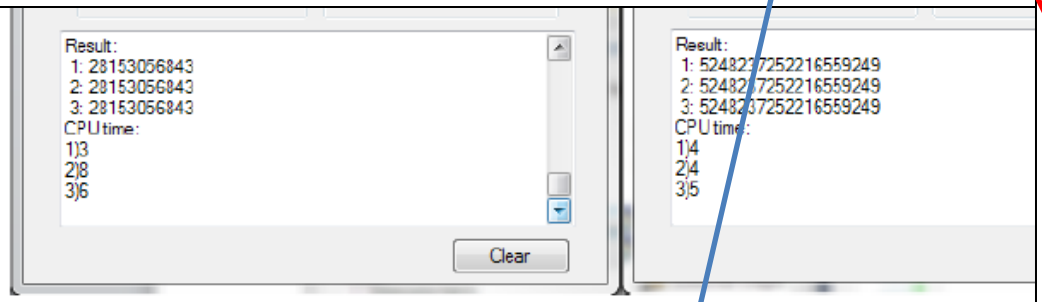
(From slides, Session 6):
Example func3:

Forgot to mention last class…

```
double func3(x,n)
{
  if n==1
     return 1
  if  (n%2)==0
     return func3(x*x,n/2)
  else
     return x*func3(x,n-1)
}
```

You got extra points, for example in HW 6-3: *if you discover any error in pseudo-code or slides.*

Stats:
- 3 noted explicitly the typo
- 7 corrected the typo
- 3 didn't notice



➔ There was a something wrong with function3: if (n==1) return 1;

```
private: double function3 (int x, int n) {
                    if (n==1) return x;

                    if ((n%2)==0)
                          return function3(x*x,n/2);
                    else return x*function3(x,n-1);
          }
```

```
private double func2(int x, int n)
{
    if (n == 1)
    {
        return x;
    }
    else
    {
        return x * func2(x, n - 1);
    }
}

private double func3(int x, int n)
{
    if (n == 1)
    {
        return 1;
    }
    if ((n % 2) == 0)
    {
        return func3(x * x, n / 2);
    }
```

Review of HWs7… Punctuation?

- HW7-1: Insertion-Sort (1pt)
- HW7-2: Merge-Sort (1pt)
- HW7-3: Compare CPU-Times (1pt)

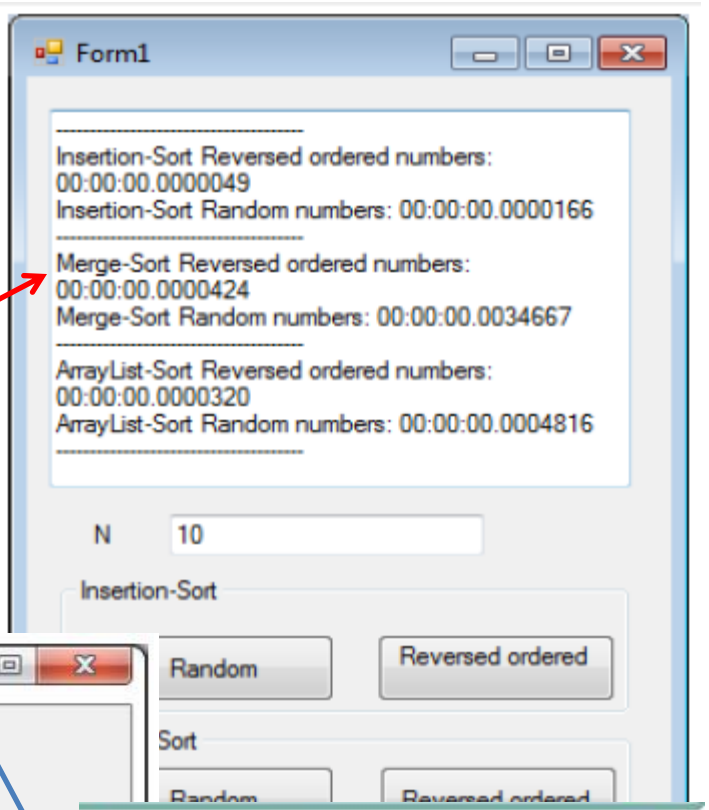
Review of HW7

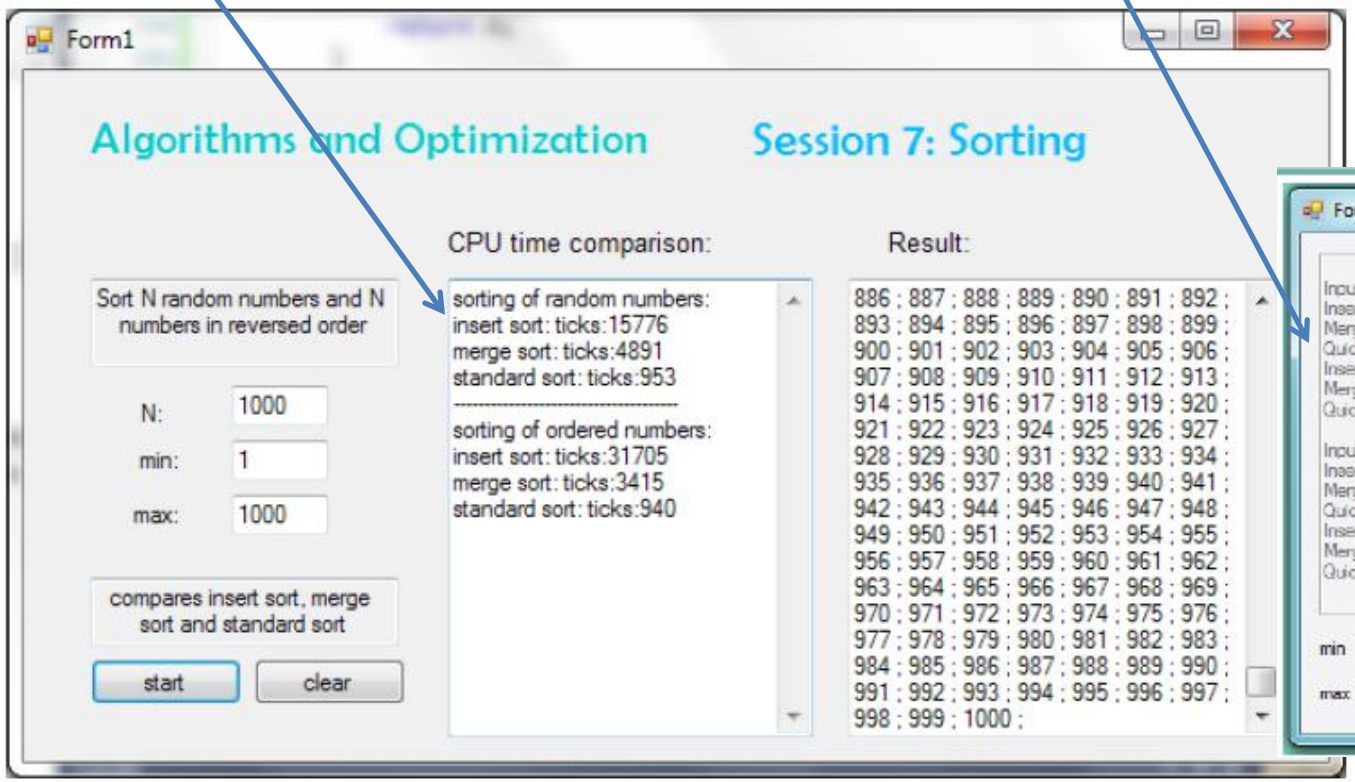
HW7-3: Compare CPU-Times

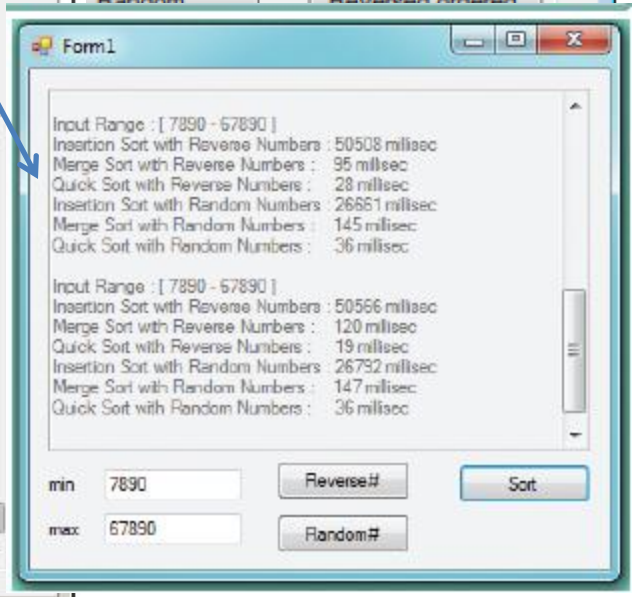
What do we expect? Insertion-sort has a complexity order of $O(n^2)$, so it should take much more CPU-time than Merge-Sort or Quicksort, which are O(n log n). However, this HW shows the contrary! Why? Because N010 is too small to show how the size of the problem influences the cpu-time. My tip: Increase N!

This is, what I wanted to see.

CPU-Time output only, is/was ok, but next time show/test also the results!







Review of Quicksort

Pseudo-code:

global int A[]={4,2,6,3,1,7,5}
Quicksort(0,6);

```
void Quicksort(m,n)
{
  if (m<n)
  {
    i=Partition(m,n)
    Quicksort(m,i-1)
    Quicksort(i+1,n)
  }
}
```

```
int Partition(m,n)
{
  v=A[m]
  i=m+1
  j=n
  while(i<=j)
  {
    while(A[i]<=v)
          i++
    while(A[j]>v)
          j—
    if (i<j){
    swap(A[i],A[j])
    i++; j—;
  }
 i--;
 swap(v,A[i])
 return i;
}
```

```
i=Partition(0,6)
{
    v=4; i=1; j=6
    while(i<=j)
    {
      while(A[i]<=4)
          i=2
      while(A[j]>4)
          j=5;j=4
      if (i=2<j=4){
      swap(A[2]=6,A[4]=1)
      ->A={4,2,1,3,6,7,5}
      i=3;j=3
      }
      while(A[i]=3<=4)
         i=4
      while(A[j]>4
    }
  i=3;
  swap(v=A[0]=4,A[3]=3)
 ->A={3,2,1,4,6,7,5}
}
```

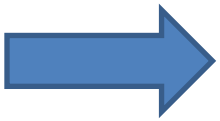
```
Quicksort(0,3)
  v=3
  while(..)
      i=3,j=3
  i=2; swap(v,A[2]=1)
  ->A={1,2,3,4,6,7,5}
Quicksort(4,6)
  v=6
  while(..){
      i=5,j=6
      if (i<j){
       swap(A[5]=7,A[6]=5)
      ->A={1,2,3,4,6,5,7}
  }
  i=5; swap(v,A[5]=5)
  ->A={1,2,3,4,5,6,7}
```

Algorithms for Searching

Sequential or Linear Search

Input:
int A[]={4,2,6,3,1,7,5}
findval=3

```
int SeqFind(A,findVal)
{
   for(i=0;i<n;i++)
   {
     if(A[i]==val)
       return i
   }
  return -1;
}
```



Order:
Best-case -> Big-Omega-> **Ω(1)**
Average-case-> Big-Theta-> **θ(n/2)**
Worst-case-> Big-O-> **O(n)**

Algorithms for Searching

Binary Search example

Example:
A={2,3,5,6,13,15,19,23 }
findVal=13

```
inf=0, sup=7
While(inf<sup)
{
   mid=3
   if A[3]=6>13
   else
     inf=3
  … while cond. 3<7
   mid=5
   if A[5]>13
     sup=5
  … while cond. 3<5
   mid=4
   if A[4]>13
   else
   …
}
```

Order:
Note that at every while-iteration, we are splitting the search in half the length of the array, then in half the half of the length of the array, etc:
n, n/2, n/4, n/8,..
Thus, the order of the algorithm is:

$n/2^k=1$ -> **O(log n)**

Note that for best-case and average case the order is also:

Ω(log n), θ(Ω(log n)

Binary Search algorithm

Input: Sorted array int[] A;

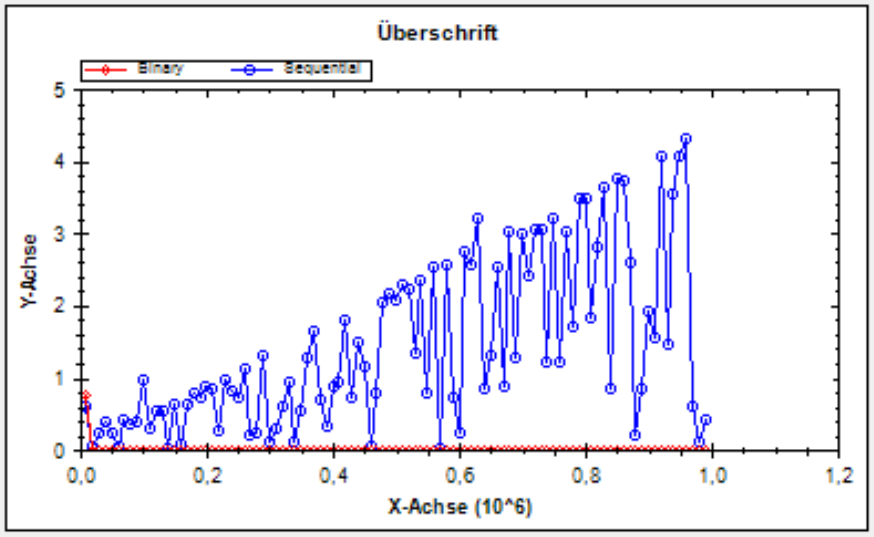
```
int BinFind(int[] A,int findVal)
{
   int inf=0;
   int sup=A.Length-1;

   while(inf<sup)
   {
     mid=(inf+sup)/2
     if (A[mid]>findval)
          sup=mid
     else
          inf=mid
   }
   if (A[inf]==findVal)
        return inf
    else
         return -1
   }
}
```

Algorithms for Searching

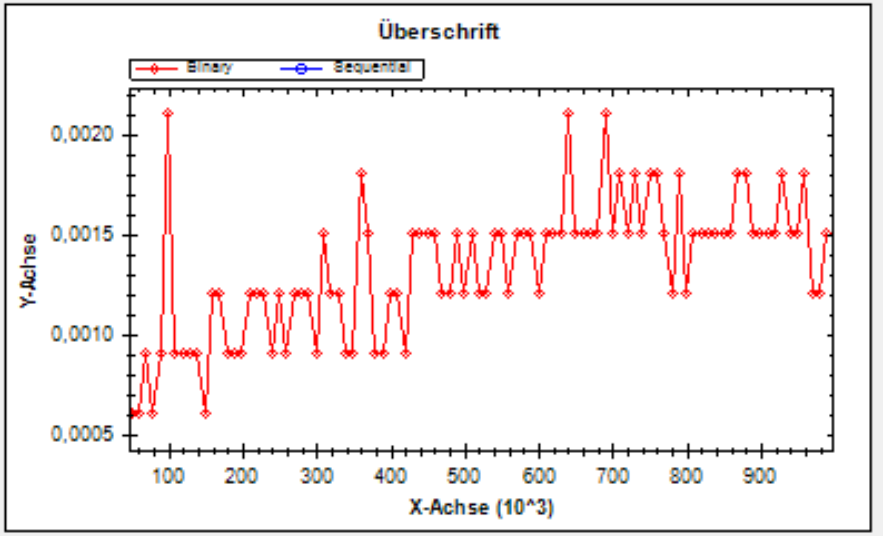
HW9-1: *Implement the Sequential and Binary Search algorithms. Note that there is a bug in the code, you may have to debug it to find it.*
HW9-2: *Show that Sequential Search is O(n) and Binary Search is O(log n). For this, plot the CPU-time for different input size.*

Examples for HW 9-2

Linear: O(n)



Logarithmic: O(log N)



If available, use chart implementation from Visual Studio:
http://archive.msdn.microsoft.com/mschart
If not, you may want to use Zedgraph:
http://sourceforge.net/projects/zedgraph/
http://www.codeproject.com/KB/graphics/zedgraph.aspx

Two code snippets that may help you

```
int minSize = 10000;
int maxSize=1000000;
Random rnd = new Random();

Stopwatch timer = Stopwatch.StartNew();
PointPairList timerTicks = new PointPairList();
PointPairList timerTicks2= new PointPairList();

for (int size = minSize; size < maxSize; size = size + 10000)
{
   input = new List<int>(size);

   for (int i = 0; i < size; i++)
      input.Add(rnd.Next(size));

   input.Sort();

   int[] inputSearch = input.ToArray();

   int findVal = rnd.Next(inputSearch.Length);

   timer.Reset();
   timer.Start();
   binFind(inputSearch, findVal);

   timer.Stop();

   timerTicks.Add(Convert.ToDouble(size), Convert.ToDouble(timer.Elapsed.Ticks / 10000.0));

   timer.Reset();
   timer.Start();
   seqFind(inputSearch, findVal);

   timer.Stop();

   list2.Add(Convert.ToDouble(size), Convert.ToDouble(timer.Elapsed.Ticks / 10000.0));
```

```
//plotting

        GraphPane myPane =
zedGraphControl1.GraphPane;
        myPane.CurveList.Clear();

        LineItem myCurve =
myPane.AddCurve("Binary",
              timerTicks, Color.Red,
SymbolType.Diamond);

        LineItem myCurve2 =
myPane.AddCurve("Sequential",
             list2, Color.Blue, SymbolType.Circle);

        zedGraphControl1.AxisChange();
```