CS3343/3341
Analysis of Algorithms
Spring 2012
Weird Topic

Test of Searches: Binary, Ternary, Fibonacci

Three Searches: Binary, Ternary, Fibonacci.

Binary Search: This search is studied in beginning programming courses. Even an unsuccessful search has only $\mathbf{log_2(n)}$ iterations, and successful searches are often quicker. [For more information on binary search see binary search.]

Ternary Search: Here the idea is to mimic binary search, but to divide the interval each time into thirds. In the code below, I managed to do this using only one **mid** variable. This search will have $\mathbf{log_3(n)}$ iterations -- an improvement, but each iteration is more complex and will take more time on the average. I expected this search to be slower than binary.

Fibonacci Search: Knuth presents a bizarre version of binary search in his Volume 3. This search doesn't divide by 2, but instead uses properties of Fibonacci numbers, namely that they can be used to divide an interval into successively smaller unequal parts, since the difference of two successive Fibonacci numbers is another one. It's amazing that it works at all. Knuth did a mathematical analysis that predicted better average run time for Fibonacci search than for the binary version. This analysis applies to Knuth's own pseudo machine language.

I copied Knuth's algorithm as he presented it, without adapting it to run with an input size different from a Fibonacci number (which can be done).

Testing The Searches.

I wanted to compare the times of binary, ternary, and Fibonacci searches. This test generated 5702887 doubles in an array, where 5702887 is the 34th Fibonacci number. The program searched for each number in the array using each of the three searches. All these searches were successful. The test was repeated 5 times.

To create unsuccessful searches I just searched for a random double. Here again there were 5702887 searches for each of the three types of search. None of these tests were successful. This test was also repeated 5 times.

The table below summarizes results from 30 runs of 5702887 searches:

5702887 searches for each entry repeated five times		
Type of search	**Successful search times**	**Unsuccessful search times**
Binary	2.26 - 2.27	9.49 - 9.54
Ternary	3.75 - 3.76	11.21 - 11.25
Fibonacci	1.95 - 2.01	9.21 - 9.31

Summarizing, Fibonacci is faster than binary, which is faster than ternary. Successful searches are 3 to 4 times faster than unsuccessful ones.

Timings for Binary and Ternary Search in Java

```
//BinSearch.java: binary, ternary, Fib. searches
//  To fit Fib. search, all searches start at 1
public class BinSearch {
```

```
// BinSearchTest.java: test 3 searches
import java.util.*;
public class BinSearchTest {
   private double[] A;
   private int N;  // N+1 a Fib. number, F[m+1]
                   //also size of array
   private int Np; // Fibonacci number, F[m]

   public BinSearch(double[] Ap, int n, int np){
      A = Ap;
      N = n;
      Np = np;
   }

   public int binarySearch(double y) {
      int low = 1;
      int high = A.length - 1;
      while (low <= high) {
         int mid = (low + high)/2;
         if (A[mid] == y) return mid; // found
         else if (A[mid] < y) low = mid + 1;
         else high = mid - 1;
      }
      return -1; // not found
   }

   public int ternarySearch(double y) {
      int low = 1;
      int high = A.length - 1;
      while (low <= high) {
         int mid = (2*low + high)/3;
         if (A[mid] == y) return mid; //found
         else if (A[mid] > y) high = mid - 1;
         else {
            low = mid + 1;
            mid = (low + high)/2;
            if (A[mid] == y) return mid;// found
            else if (A[mid] > y) high = mid -1;
            else low = mid + 1;
         }
      }
      return -1; // not found
   }

   public int fibonacciSearch(double y) {
      // Fibonacci search, search A[1],...,A[N]
      // Here N+1 must be a Fib. number, F[m+1]
      int mid = Np;             //   F[m]
      int p     = (N+1) - Np;   //   F[m-1]
      int q     = 2*Np - (N+1); //   F[m-2]
      for (;;) {
         if (y == A[mid]) return mid; // found
         else if (y < A[mid]) {
            if (q == 0) return -(mid - 1);// not
            mid = mid - q;
            int temp = p;
            p = q;
            q = temp - q;
         }
         else if (y > A[mid]) {
            if (p == 1) return -mid;// not found
            mid = mid + q;
            p = p - q;
            q = q - p;
         }
      }
   }
}
```

```
   private double[] A;
   private Quicksort quick;
   private BinSearch bin;
   private Random r = new Random(31415);

   public BinSearchTest(int N, int Np){
      // N+1 = F[m+1], Np = F[m], Fib #s
      A = new double[N+1];
      quick = new Quicksort(A);
      bin = new BinSearch(A, N, Np);
   }

   public void randomTest() {
      long startTime;
      startTime =  System.currentTimeMillis();
      for (int i = 0; i < A.length; i++)
         A[i] = r.nextDouble();
      elapsedTime(startTime, "Time to generate, ");

      startTime =  System.currentTimeMillis();
      quick.quicksort();
      elapsedTime(startTime, "Time to sort, ");
      System.out.println("isSorted: " +quick.isSorted());

      startTime =  System.currentTimeMillis();
      for (int i = 1; i < A.length; i++) {
         int i1 = bin.binarySearch(A[i]);
         if (i1 != i)
            System.out.print("(" + i + "," + i1 + "),");
      }
      elapsedTime(startTime, "After binary    search, ");

      startTime =  System.currentTimeMillis();
      for (int i = 1; i < A.length; i++) {
         int i2 = bin.ternarySearch(A[i]);
         if (i2 != i)
            System.out.print("(" + i + "," + i2 + "),");
      }
      elapsedTime(startTime, "After ternary   search, ");

      startTime =  System.currentTimeMillis();
      for (int i = 1; i < A.length; i++) {
         int i1 = bin.fibonacciSearch(A[i]);
         if (i1 != i)
            System.out.print("(" + i + "," + i1 + "),");
      }
      elapsedTime(startTime, "After Fibonacci search, ");
   }

   public static void elapsedTime(long startTime,
         String s) {
      long stopTime = System.currentTimeMillis();
      double elapsedTime =
         ((double)(stopTime - startTime))/1000.0;
      System.out.println(s + "elapsed time: " +
         elapsedTime + " seconds");
   }

   public static void main(String[] args) {
      //final int N = 88; // N+1 = Fib # F[m+1]
      //final int Np = 55; // = Fib # F[m]
      final int N = 5702886; // N+1 = Fib # F[m+1]
      final int Np = 3524578; // = Fib # F[m]
      BinSearchTest bin = new BinSearchTest(N, Np);
      bin.randomTest();
   }
}
```

Output. Notice that variables p, and q below right are always Fibonacci numbers.

```
Typical output (sucsessful searchs):
Time to generate, elapsed time: 1.538 seconds
Time to sort, elapsed time: 2.263 seconds
isSorted: true
After binary    search, elapsed time: 2.257 sec
```

```
Debug output from the Fibonacci search, showing
values of the variables i, p, and q at each iteration.
Notice that these are all Fibonacci numbers.

Using N = 88, Np = 55, searching for 20, got:
```

```
After ternary   search, elapsed time: 3.743 sec
After Fibonacci search, elapsed time: 2.002 sec
```

```
Time to generate, elapsed time: 1.581 sec
Time to sort, elapsed time: 2.223 seconds
isSorted: true
After binary    search, elapsed time: 2.253 sec
After ternary   search, elapsed time: 3.751 secs
After Fibonacci search, elapsed time: 1.986 sec
```

Typical output (unsucsessful searchs):

```
Time to generate, elapsed time: 1.54 seconds
Time to sort, elapsed time: 2.224 seconds
isSorted: true
After binary    search, elapsed time: 9.528 sec
After ternary   search, elapsed time: 11.248 sec
After Fibonacci search, elapsed time: 9.23 sec
```

```
Time to generate, elapsed time: 1.793 sec
Time to sort, elapsed time: 2.275 sec
isSorted: true
After binary    search, elapsed time: 9.536 sec
After ternary   search, elapsed time: 11.244 sec
After Fibonacci search, elapsed time: 9.305 sec
```

```
i:55    p:34    q:21
i:34    p:21    q:13
i:21    p:13    q:8
i:13    p:8     q:5
i:18    p:3     q:2
i:20    p:1     q:1
```

```
Using N = 5702886, Np = 3524578, searching for 2000000:
i:3524578   p:2178309   q:1346269
i:2178309   p:1346269   q:832040
i:1346269   p:832040    q:514229
i:1860498   p:317811    q:196418
i:2056916   p:121393    q:75025
i:1981891   p:75025     q:46368
i:2028259   p:28657     q:17711
i:2010548   p:17711     q:10946
i:1999602   p:10946     q:6765
i:2006367   p:4181      q:2584
i:2003783   p:2584      q:1597
i:2002186   p:1597      q:987
i:2001199   p:987       q:610
i:2000589   p:610       q:377
i:2000212   p:377       q:233
i:1999979   p:233       q:144
i:2000123   p:89        q:55
i:2000068   p:55        q:34
i:2000034   p:34        q:21
i:2000013   p:21        q:13
i:2000000   p:13        q:8
```

Revision date: **2011-11-03**. (Please use **ISO 8601**, the International Standard Date and Time Notation.)
