Report: Linear Search Performance Comparison

Yu Sheng

2018/2/7

The main purpose of study is to compare the performance of each search using the execution time data of four searches including linear search, better search sentinel linear search and recursive linear search. This comparison process will be divided into three parts; first part is analyzing the time duration curve of each search algorithm in each case: best case, worst case and average case. We are using C language for implementing the algorithm in this part. Second part is to compare the speed of worst case of each search program in Java and C. Last but not least, we will look closely to the number of searches in one second for each search program in average case.

Before going to the data analyzing part, I will illustrate how we obtain the data first. The first step is to implement each search program using the algorithm provided in the lab#2 instruction sheet. Then insert time command to count the time duration of each search program. In order to make observation easier, I choose millisecond as unit for duration records for both C and Java. Call element we are searching as x and the array of data where we search x as A. Be carefully to shuffle array each time. Set x to the value of first element in A for the best case x. And for average case, set x to a random number in bigger range than range of A. and x=-1 in worst case, since -1 would never occur in the A. Then increase the data size from 1 million to 50 million by 1 million each time and do 100 bootstraps for each data size and averaging them, we will obtain an unbiased dataset of time duration for each search program on data size from 1 million to 50 million in best, average and worst case.

Then we for the first part of this report, we would see the performance graph of four linear search algorithm in three cases using C language and analysis the individual trend and the difference between four searches. And also compare to the theoretical O value of each algorithm. For all graphs, y-axis represents time duration in milliseconds and x- axis represents the data size from 1 million to 50 million increments by 1 million.

Fig 1.1 is the time versus data size graph for Best Case, in other word, we will find target element in the first position in array. For the linear search, we can see a straight upward blue line in the graph. The time duration of linear search is increasing approximately linearly as data size increases. It ranges from around 0 to over 100 milliseconds. The reason behind it is in the linear search algorithm. In linear search algorithm, we find we will always search the whole array even if we already find the element. Thus the time duration for search is increasing since we need search more elements when the data size gets larger. But when we looking at better search, sentinel search and recursive search curves, we find three lines are almost overlapped. They are parallel to x-axis and their y is constantly close to zero. Thus we conclude that the time duration does for these three searches do not change significantly according to the increases of data size, instead it stays extremely small for all time. And this phenomenon makes sense if we look into the algorithm of these three searches. For all the three, it stops right after the target element x is found. And since we are in the best case and we will find the x in first loop, then whole program stops immediately. And it happens no matter how large the data size is, thus the time taken to run these search program are extremely short constantly. This result perfectly matches the expected value from the textbook, as in best case, linear search has O(n^2) complexity but the time complexity for better, sentinel and recursive search are all constant.

Fig 1.1

**Worst Case**

Fig 1.2

Fig 1.2 is the time versus data size graph for Average Case

we will compare the result with our theoretical big O data of each search algorithm.

Fig 1

Appendix of Codes

//Yu Sheng

//2018/2/6

//In C: performance of the search

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int linearSearch(int data[],int n, int x) {

 int result=-1;

 for(int i=0;i<n;i++){

if(data[i]==x){

  result=i;

}//if()

 }//for()

 return result;

}//linearSearch()

int betterSearch(int data[],int n, int x) {

 for(int i=0;i<n;i++){

if(data[i]==x){

return i;

}//if()

 }//for()

 return -1;

}//betterSearch()

int sentinelLinearSearch(int data[],int n, int x) {

 int Last=data[n-1];

 data[n-1]=x;

 int i=0;

 while(data[i]!=x){

i++;

 }//while

 data[n-1]=Last;

 if(i<n-1||data[n-1]==x)

 return i;

 else

 return -1;

}//sentinelLinearSearch()

int recursiveLinearSearch(int data[],int n, int i,int x) {

 if(i>=n)

return -1;

 else if (data[i]==x)

return i;

 else

return recursiveLinearSearch(data,n, i+1,x);

}//recursiveLinearSearch()

int main(void){

 long start;

 int ls,bs,sls,rls;

int size;

int A[size];

for(int i=0;i<size;i++){

  A[i]=i;

}//for()

 for(int i=0;i<size;i++){

int j=rand()%(size-i)+i;

int swap = A[i];

A[i]=A[j];

A[j]=swap;

 }//shuffle

//  for(int b=0;b<10;b++){

  //int x=rand()%(size\*2);

  int x=A[0];

  //int x=-1;

  start=clock();

  ls=linearSearch(A,size,x);

  double d1=(double)(clock()-start)/(float)CLOCKS\_PER\_SEC\*1000.0;

  start=clock();

  bs=betterSearch(A,size,x);

  double d2=(double)(clock()-start)/(float)CLOCKS\_PER\_SEC\*1000.0;

  start=clock();

  sls=sentinelLinearSearch(A,size,x);

  double d3=(double)(clock()-start)/(float)CLOCKS\_PER\_SEC\*1000.0;

  start=clock();

  rls=recursiveLinearSearch(A,size,0,x);

  double d4=(double)(clock()-start)/(float)CLOCKS\_PER\_SEC\*1000.0;

printf("%f %f %f %f\n",d1,d2,d3,d4);

 }//data size

  return 0;

}

//Yu Sheng

//2018/2/6

//In JAVA: performance of the search

import java.util.Random;

public class search {

   public static void main(String[] args) {

   int ls,bs,sls,rls;

   double d1,d2,d3,d4;

   long start;

   int size=1000000;

   int [] A= new int[size];

   Random rand=new Random();

   for(int i=0;i<size;i++){

   A[i]=i;

   }//initialize array

   for(int i=0;i<size;i++){

   int j=rand.nextInt(size-i)+i;

   int swap = A[i];

   A[i]=A[j];

   A[j]=swap;

   }    //shuffle

     int x=rand.nextInt(size\*2);

   // int x=A[0];

   //int x=-10;

   start =System.nanoTime();

ls=linearSearch(A,size,x);

d1= (System.nanoTime()-start)/1000000.0;

   start =System.nanoTime();

   bs=betterSearch(A,size,x);

   d2= (System.nanoTime()-start)/1000000.0;

   start =System.nanoTime();

   sls=sentinelLinearSearch(A,size,x);

   d3=(System.nanoTime()-start)/1000000.0;

   start =System.nanoTime();

   rls=recursiveLinearSearch(A,size,0,x);

   d4= (System.nanoTime()-start)/1000000.0;

   System.out.println(d1+" "+d2+" "+d3+" "+d4);

}//from 1m to 50m

}//main()

 public static int linearSearch(int[] data,int n, int x) {

     int result=-1;

for(int i=0;i<n;i++){

  if(data[i]==x){

    result=i;

  }//if()

}//for()

return result;

 }//linearSearch()

 public static int betterSearch(int[] data,int n, int x) {

for(int i=0;i<n;i++){

  if(data[i]==x){

  return i;

  }//if()

}//for()

return -1;

 }//betterSearch()

 public static int sentinelLinearSearch(int[] data,int n, int x) {

int Last=data[n-1];

data[n-1]=x;

int i=0;

while(data[i]!=x){

  i++;

}//while

data[n-1]=Last;

if(i<n-1||data[n-1]==x)

return i;

else

return -1;

 }//sentinelLinearSearch()

public static int recursiveLinearSearch(int[] data,int n, int i,int x) {

if(i>=n)

  return -1;

else if (data[i]==x)

  return i;

else

  return recursiveLinearSearch(data,n,i+1,x);

}//recursiveLinearSearch()

}