Linear Search Performance Comparison Report

Yu Sheng

2018/2/7

The main purpose of the 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; the first part is analyzing the time duration curve of each search algorithm in three cases: best case, worst case, and average case. We are using C language for implementing the algorithm in this part and we will compare it to the result of the discussion in class in Table 1. The 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 at the number of searches in one second for each search program in average case.

**Table 1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Best Case | Worst Case | Average Case | | Efficiency |
| Linear Search | O(n) | O(n) | | O(n) | 3rd efficient |
| Better Search | O(1) | O(n) | | O(n) (3n) | 2nd efficient |
| Sentinel Linear Search | O(1) | O(n) | | O(n) (2n) | 1st efficient |
| Recursive Linear Search | O(1) | O(n) | | O(n) | 4th efficient |

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 sheets. Then insert time command to count the time duration of each search program. In order to make observation easier, I choose millisecond as the unit for duration records for both C and Java. Call element we are searching for x and the array of data where we search x as A. Be careful 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 at least 20 runs for each data size and select the reasonable one, 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.

By following, we would see the performance graph of four linear search algorithms 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.

**Best case**

Fig 1.1

Fig 1.1 is the time versus data size graph for Best Case, in other words, we will find target element in the first position in the 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 around 107 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 the x-axis and their y is constantly close to zero. And this phenomenon makes more sense if we look closely into the algorithm of these three searches. For all three, they stop right after the target element x is found. And since we are in the best case, we will find the x in the first loop where whole program stops immediately. Thus we conclude that the time duration for these three searches do not change significantly according to the increment of data size, instead it remain seats below 0.01 millisecond. This result perfectly matches the expected value from the textbook as in table 1. In the best case, linear search has O(n) linear complexity but the time complexity for better, sentinel and recursive search are all constant(O(1)).

**Worst Case**

Fig 1.2.1 Fig 1.2.2

Fig 1.2.2 is the time versus data size graph for the worst case. In other words, target element will be at the end of array or not even existing in the array. Thus due to the algorithm, we will search through the whole array for each run time. Looking at the Graph 2.2.1, for the linear search, better search and sentinel search, we find linear search and better search curves are overlapped (blue and red line) and they are little above the sentinel search curve within 10 milliseconds. And time duration of these three searches is increasing linearly as data size increases. And this is reasonable because in worst case we are searching the whole array for all four types of searches; while the data size increases, there are more elements we need to search. Besides, the range of these three curves is all from 2 milliseconds to about 100 milliseconds. And these linear curve agrees with our expect ion (Table 1), where the complexity of search in worst case are all O(n). However, we notice that in Fig 1.2.2, although recursive search is also a linear increasing curve and it also has linear time complexity, the time duration of recursive is increasing in a much larger rate and the range of its time duration is 7 times larger than that of other searches. In other words, while it takes linear search program 103 milliseconds to search in 50 million data, it takes 688 milliseconds for recursive search program. And the possible reason behind it may be the creation of a stack frame while doing each recursion in recursive search. And for 50 million data, we need to create so many stack frames, which is extremely time-consuming. That is also why we need to increase the stack size limit in java using “java -Xss1000000m search” and type “ulimit -s unlimited” in Linux when process recursive search for large data set. Therefore, although the complexity of recursive linear search is the same as other three searches as in Table 1, we are getting significantly larger execution time for recursive search compare to other search algorithms.

**Average Case**

Fig 1.3.1 Fig 1.3.2

Fig 1.3.2 shows us how four search algorithms perform in average case, that is target element can be in the random position in the array or even out the range of array. We can see four positive trending curves in Fig 1.3. And there are larger fluctuations in better, sentinel and recursive search when dataset size increases, since random x will fluctuate in a larger range and therefore there are more uncertainties in choosing the random x. But looking at the overall trend, the time duration of all four searches is still in a linear relationship to the dataset size, which concord with our discussion in class, as all four searches have linear complexity in average case (Table 1). Another thing I notice is that the average value of time duration for sentinel search and better search is shorter than that of linear search and the difference becomes bigger when the data size gets larger. Because linear search always searches the whole array, but better and sentinel search stop when we are in the position of that random x. As data set becomes bigger, the time saving by stop right after found the element will be more obvious. Furthermore, compare better search with sentinel search, we can see the time duration curve for sentinel search stays constantly below the curve of better search. In other words, we could conclude that sentinel search is more efficient than better search. And according to our discussion in class where we counted that there are approximately 2n steps in sentinel but 3n steps in better search when searches an element in array size n. Thus ideally, two curves should start by close initial time duration values. But as the data size increases, time duration curve for sentinel search should increase at a lower rate than that of better search. Eventually, we should see sentinel search curve stays below the better search curve. And that is exact what happens in my graph. Due to the same reason in worst case, we can see the curve for time duration of recursive search is increasing much more rapid than other searches, and it ranges from 2 milliseconds to about 500 milliseconds.

At the end of this part, I want to spend some time addressing the general patterns of curve shape of each search in three cases. We can see the time duration curve for linear search are in the same range and same shape for three cases. That is because we are always searching through whole array for linear search algorithm. And for better search, sentinel search and recursive search, the algorithm states that once we find the element, we would stop searching. Thus the time duration for the searching process depends on the position of target value in the array, former position the target value has in the array, quicker the time duration would be. Thus if the target element is A[0], then it takes less than 0.1 milliseconds for these three searches. And the value for average curve should always between the worst case and best case value. And actually, since I set the range of random x to be twice as the range of A, it is very possible for a worst case to happen under this situation. Therefore, for better, sentinel and recursive search, the average value of time duration is about 70 % of worst case value.

Fig 3.1 Fig 3.2

Fig 3.3 Fig 3.4

Moving to the second part, we are trying to get a comparison of program speed in a different language. We are doing it under worst case where we can best distinguish the comparative speed of each searching algorithms. I got the result as shown in four graphs above where blue curve represents C and red curve represents Java. We see that curve represents program written in C has significantly larger slope than the curve of program in Java, which means Java is running much faster than C. And for linear, better and sentinel search (Fig 3.1, Fig 3.2, Fig 3.3), the speed in Java is 5 times quicker than the speed in C. However, in theory, C should run far quicker than in Java, since C runs directly and C operates at a lower level compared to Java. While testing the program again and again and getting the same result with classmates also using C, I made following assumptions about why this would possibly happen. Combining the information online, it may caused by the optimization. Java applies optimizations dynamically, but we compile C using gcc -o which optimize nothing. Another possibility is that we measure the time duration more precisely in Java, while we are measuring it in seconds in C. Thus the preciseness of the measurement is not enough for the comparison between the languages. What’s more, for the recursive research in Fig 2.4, the difference between two is not obvious. It would be reasonable then, since for the recursive research, the time duration is comparatively very large, timing command can now record the value in C more precisely, thus the time duration in two languages are very close.

For the last part of extra credits, I record the number of searches in one second in average case use while loop and my result are shown in the Table below.

**Table 2**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Data size(million) | Linear Search | | Better Search | | Sentinel Search | | Recursive Search |
| 10 | 47 | 96 | | 106 | | 18 | |
| 20 | 24 | 42 | | 51 | | 5 | |
| 30 | 16 | 30 | | 36 | | 4 | |
| 40 | 12 | 28 | | 27 | | 3 | |
| 50 | 10 | 20 | | 22 | | 3 | |

From this table, we notice that while the size of A increases, searches per second decreases. And it is clear that sentinel search is the quickest search, it performs 106 searches in one second. Better search is the second fast, Linear Search is the third and recursive search is the slowest, it only runs 18 searches, is one-fifth of the number of searches per second for sentinel search.

In conclusion, the performance of linear search stays the same for all cases. And time duration of execution of four searches increases linearly as data size increases except in best case, it always takes less than 0.01 millisecond for better, sentinel and recursive search to run. Recursive search is the lowest search algorithm due to its stack frame construction. Linear is second lowest as it searches whole array no matter in what situation. Sentinel Search would be the quickest search and better search would be second quickest combining the information from time duration versus dataset size graph in average case and searches in one-second table.

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

  //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);

  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);

}//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()

}

//Yu Sheng

//2018/2/6

//Extra credits in C: searches in 1 second in average case

#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=1000000;

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

int s=0;//number of search per second

double d1=0;

while(d1<1){

int x=rand()%(size+100);

start=clock();

//linearSearch(A,size,x);

betterSearch(A,size,x);

//sentinelLinearSearch(A,size,x);

//recursiveLinearSearch(A,size,0,x);

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

s++;

}//while in one second

printf("%d\n",s);

return 0;

}