10802EE 398000 Algorithms

Homework 2. Random Data Searches report

● Introduction

In this report, I will implement three kinds of primitive searching algorithms. Three kinds of searching algorithms are introduced below :

Algorithm	Concept
Linear search	Search the data from the beginning of the data set to the end.
Bidirection search	Search the data from the beginning and the end to the middle of the data set simultaneously in each iteration.
Random-direction search	Use rand() function to choose doing the forward search (from the beginning to the end of the dataset) or the backward search (from the end to the beginning of the dataset).
Table 1 The concepts of the three searching algorithms.	

I will also prove them and analyze their space and time complexities, then measure the average CPU time (Number of searching repetitions for a data is equal to 500) and the worst-case CPU time (Number of searching repetitions for a data is equal to 5000) of these algorithms. Noticed that assuming searches are all successful. Finally, I will show the relation between the performance and the time complexities to see whether the time complexities I derived are correct or not. Besides, I will compare the performance between three kinds of algorithms and explain the reasons.

Approach

The following are pseudocodes of three searching algorithms, their proof, and their time and space complexity analysis. I will organize their time and space complexity at the end of this section.

```
// Find the location of word in array list.
// Input: word, array list, int n
// Output: int i such that list[i] = word.
1 Algorithm search(word, list, n)
2 {
3       for i := 1 to n do { // compare all possible entries
4             if (list[i] = word) return i;
5       }
6       return −1; // unsuccessful search
7 }
```

Algorithm 1 Linear search

Theorem 1 Algorithm search(word, list, n) correctly search the word from the list[1 : n] of $n \geqq 1$ elements. If the word is found, then return the index j such that list[j] is equal to the word, if the word cannot be found then -1 is returned.

Proof: Assuming the word is at index j where $n \geqq j \geqq 1$, the execution of lines 3 to 5 will run through the list from i = 1 to i = j, then returns j which is the correct answer. If the word is not in the list, lines 3 to 5 will run through the whole list, then returns -1 implying the searching is failed.□

The space complexity of Algorithm search(word, list, n) is equal to O(n) because it is characterized by n, the number of elements in the list.

According to **Table 2**, the time complexity of Algorithm search(word, list, n) is equal to 2n (worst-case) and 2 (best-case). Besides, it seems that the worst-case time complexity is O(n) and the best-case time complexity is Ω(1) because the asymptotic time complexity is determined by the inner most loop step, which is n and 1 respectively. How about the average time complexity? Assuming that the list[1 : n] has n elements, the average time complexity will equal to (2 + 4 + 6 + … + 2n) / n = n + 1 which is Θ(n).

statement	s/e	Frequency		total steps	
		worst case	best case	worst	best
1 Algorithm search(word, list, n)	0	--	--	0	0
2 {	0	--	--	0	0
3 for i := 1 to n do {	1	n	1	n	1
4 if (list[i] = word) return i;	1	n	1	n	1
5 }	0	--	--	0	0
6 return −1; // unsuccessful search	1	0	0	0	0
7 }	0	0	0	0	0
Total				2n	2

Table 2 Step table of Linear search (Assuming searches are all successful.)

```
// Bidirectional search to find the location of word in array list.
// Input: word, array list, int n
// Output: int i such that list[i] = word.
1 Algorithm BDsearch(word, list, n)
2 {
3       for i := 1 to n/2 do { // compare all entries from both directions
4             if (list[i] = word) return i;
5             if (list[n – i – 1] = word) return n – i – 1;
6       }
7       return −1; // unsuccessful search
8 }
```

Algorithm 2 Bidirection search

Theorem 2 Algorithm BDsearch(word, list, n) correctly search the word from the list[1 : n] of $n \geqq 1$ elements. If the word is found, then return the index j such that list[j] equals to the word, if the word cannot be found then -1 is returned.

Proof: Assuming the word is at index j where $n \geqq j \geqq 1$ and the index k is the element currently visited. The execution of lines 3 to 6 will run through the list from k = 1 and k = n – i – 1 both sides to the middle of the list, and stop at k = i = j (if $j \leqq n / 2$) or k = n – i – 1 = j (if $j > n / 2$) then returns j which is the correct answer. If the word is not in the list, lines 3 to 6 will scan through the whole

list, then returns -1 implying the searching is failed.□

The space complexity of Algorithm BDsearch (word, list, n) is equal to O(n) because it is characterized by n, the number of elements in the list.

According to **Table 3**, the time complexity of Algorithm BDsearch is equal to 3n / 2 (worst case) and 2 (best case). We can also know that the asymptotic time complexity is equal to O(n) and Ω(1) respectively. The average time complexity is equal to ((2 + 4 + … + n) * 2) / n = n + 2 which is Θ(n).

statement	s/e	Frequency		total steps	
		worst	best	worst	best
1 Algorithm BDsearch(word, list, n)	0	--	--	0	0
2 {	0	--	--	0	0
3 for i := 1 to n/2 do {	1	n / 2	1	n / 2	1
4 if (list[i] = word) return i;	1	n / 2	1	n / 2	1
5 if (list[n – i + 1] = word) return n – i + 1;	1	n / 2	0	n / 2	0
6 }	0	0	0	0	0
7 return −1; // unsuccessful search	0	0	0	0	0
8 }					
Total				3n / 2	2

Table 3 Step table of Bidirection search (Assuming searches are all successful.)

```
// Random-direction search to find the location of word in array list.
// Input: word, array list, int n
// Output: int i such that list[i] = word.
1 Algorithm RDsearch(word, list, n)
2 {
3       choose j randomly from the set {0, 1} ;
4       if (j = 1) then
5             for i := 1 to n do { // forward search
6                   if (list[i] = word) return i;
7             }
8       else
9             for i := n to 1 step −1 do { // backward search
10                  if (list[i] = word) return i;
11            }
12      return −1; // unsuccessful search
13 }
```

Algorithm 3 Random-direction search

Theorem 3 Algorithm RDsearch(word, list, n) correctly search the word from the list[1 : n] of n ≧ 1 elements. If the word is found, then return the index j such that list[j] equals to the word, if the word cannot be found then -1 is returned.

Proof: Assuming the word is at index k where n ≧ k ≧ 1. If j = 1, then the program runs line 5 to 7, which is same as Linear search. It has already been proven in **Theorem 1**. If j = 0, then the program r runs line 9 to 11, which is just the reverse version of Linear search. It can also be proved by **Theorem 1** with changing traveling direction. □

The space complexity of Algorithm RDsearch (word, list, n) is equal to O(n) because it is characterized by n, the number of elements in the list.

According to **Table 3**, the average time complexity of Algorithm RDsearch is equal to n when n is sufficiently large. This is because the probability of j equal 1 is same as the probability of j equal 0 when n is sufficiently large. We can also know that the asymptotic average time complexity is equal to Θ(n).

Besides, from **Table 3**, it seems that the time complexity is independent of k (the index of the word). We can come to the conclusion that the best-case time complexity, the worst-case time complexity, and the average time are the same equal to n. This is the reason why I pick word = list[n / 2] for the worst-case CPU time evaluation in the program. I can also pick list[1], list[2], …, list[n] anyone for the worst-case CPU time evaluation either.

statement	s/e	frequency		total steps
		j = 0	j = 1	
1 Algorithm RDsearch(word, list, n)	0	--	--	0
2 {	0	--	--	0
3 choose j randomly from the set {0, 1} ;	1	1	1	1
4 if (j = 1) then	1	1	1	1
5 for i := 1 to n do {	1	k	0	(k / 2) + 0 = k / 2
6 if (list[i] = word) return i;	1	k	0	(k / 2) + 0 = k / 2
7 }	0	0	0	0
8 else	0	0	0	0
9 for i := n to 1 step −1 do {	1	0	n-k	0 + (n – k) / 2 = (n – k) / 2
10 if (list[i] = word) return i;	1	0	n-k	0 + (n – k) / 2 = (n – k) / 2
11 }	0	0	0	0
12 return −1;	1	0	0	0
13 }	0	0	0	0
Total				n

Table 4 Step table of Random-direction search (Assuming searches are all successful.)

Noticed that k is the index of word and the probability of j = 0 is equal to j = 1 which is 1 / 2.

	Space complexity	Time complexity					
		Best-case		Worst-case		Average-case	
Linear search	O(n)	Ω(1)	2	O(n)	2n	Θ(n)	n+1
Bidirection search	O(n)	Ω(1)	2	O(n)	3n / 2	Θ(n)	n+2
Random-direction search	O(n)	Ω(n)	n	O(n)	n	Θ(n)	n

Table 5 the space and time complexities of three algorithms

- **Results and Observation**

n	Average CPU time			Worst-case CPU time		
	Linear	**Bidirection**	**Random**	**Linear**	**Bidirection**	**Random**
10	7.5960E-08	6.73771E-08	9.84192E-08	1.1540E-07	1.1301E-07	1.09386E-07
20	1.2510E-07	1.18804E-07	1.52302E-07	2.2416E-07	2.15197E-07	1.51968E-07
40	2.3021E-07	2.21395E-07	2.61593E-07	4.5738E-07	4.25005E-07	2.62165E-07
80	4.6135E-07	4.2817E-07	4.64153E-07	8.5383E-07	7.73191E-07	4.42982E-07
160	8.3959E-07	6.29064E-07	6.29836E-07	1.2084E-06	1.12462E-06	6.30045E-07
320	1.3348E-06	1.12023E-06	1.18651E-06	2.435E-06	2.23141E-06	1.18923E-06
640	2.4068E-06	2.22965E-06	2.34092E-06	4.6534E-06	4.42963E-06	2.32759E-06
1280	4.7708E-06	4.53466E-06	4.73539E-06	9.3992E-06	9.21278E-06	4.68721E-06
2560	9.4412E-06	9.15808E-06	9.43152E-06	1.8826E-05	1.83758E-05	9.43422E-06

Table 6 CPU time of three kinds of searching algorithm

From **Table 6**, it seems that when n is double, the CPU time for each case is also double. The result consists with **Table 5,** indicating that the time complexities of three kinds of algorithms are all O(n) and Θ(n) for both the worst-case and the average case.

The following figures are the results of three algorithms:

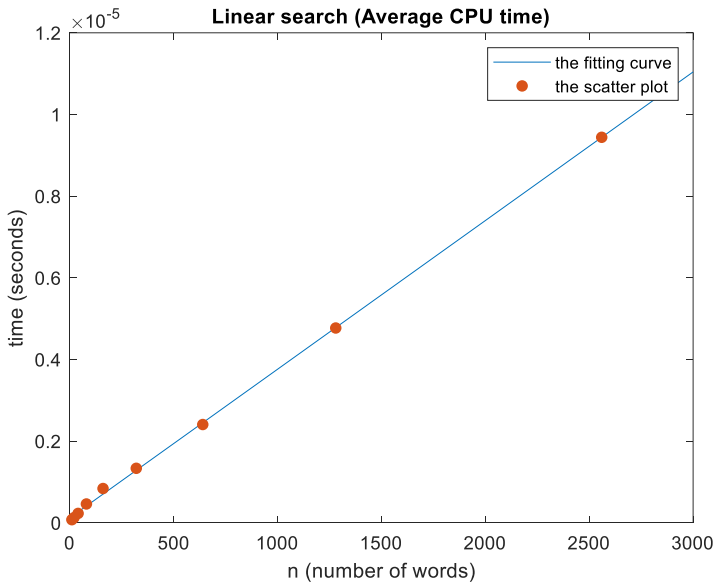

Figure 1 Linear search result (Average). The result of the average CPU time vs. n (the number of words) is linear. The result consists with the time complexity Θ(n) which I analyzed before.

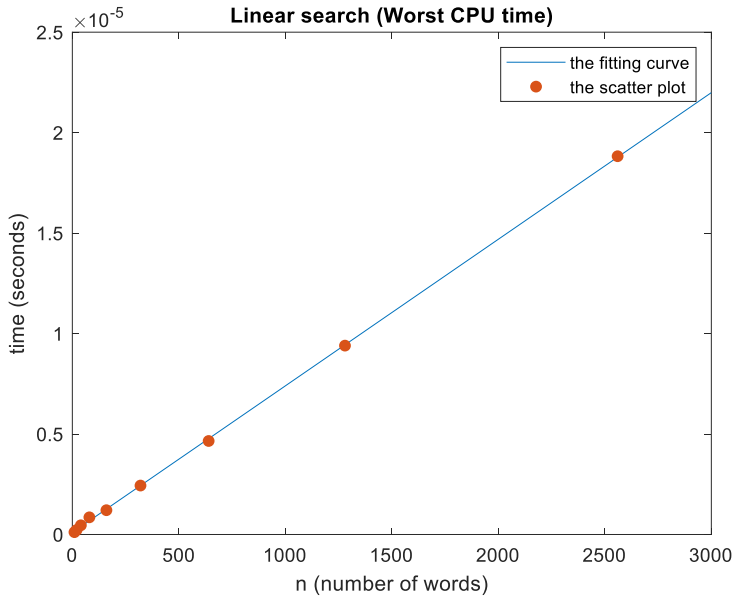

Figure 2 Linear search result (Worst). The result of the worst-case CPU time vs. n (the number of words) is linear. The result consists with the time complexity O(n) which I analyzed before.

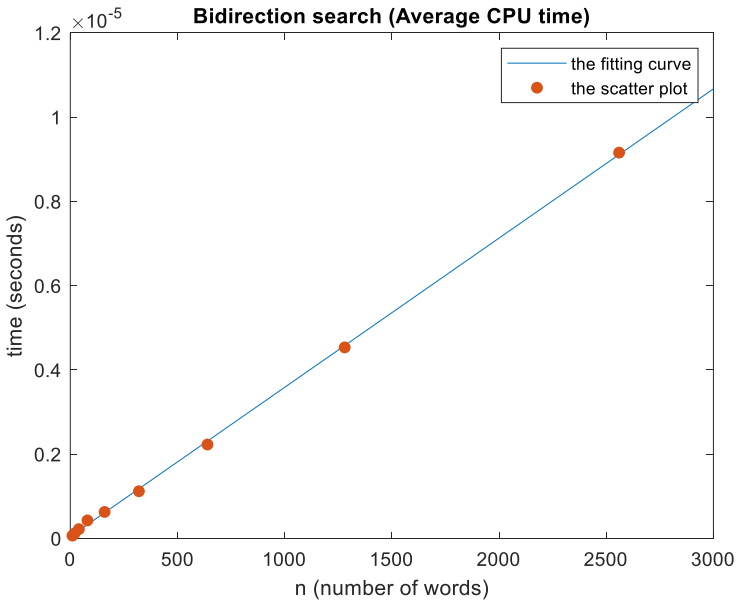

Figure 3 Bidirection search result (Average). The result of the average CPU time vs. n is linear. The result consists with the time complexity Θ(n) which I analyzed before.

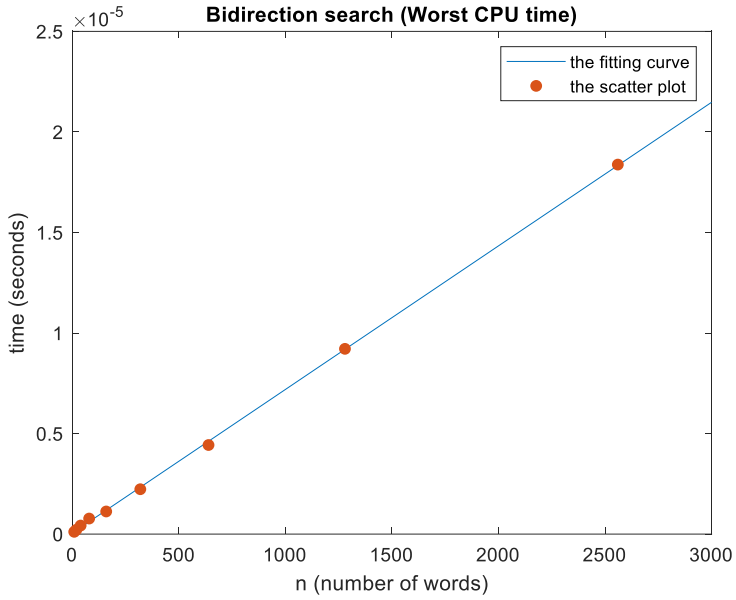

Figure 4 Bidirection search result (Worst). The result of the worst-case CPU time vs. n is linear.

The result consists with the time complexity O(n) which I analyzed before.

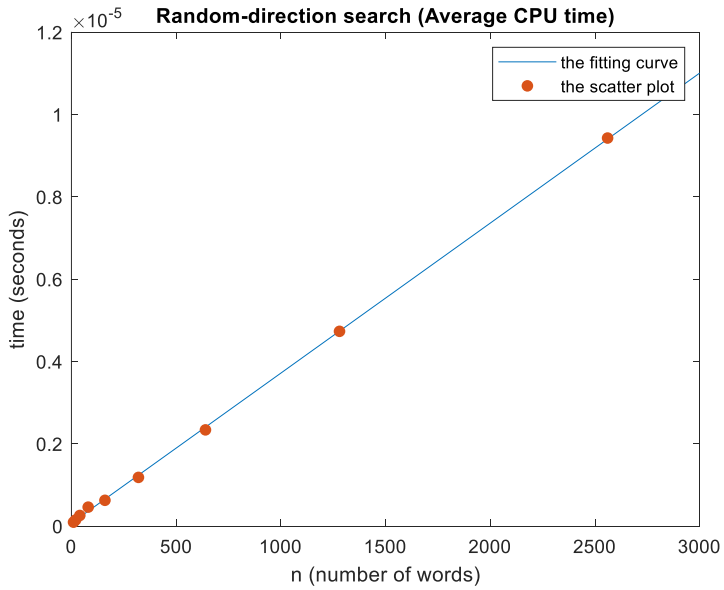

Figure 5 Random-direction search result (Average). The result of the average CPU time vs. n is linear. The result consists with the time complexity Θ(n) which I analyzed before.

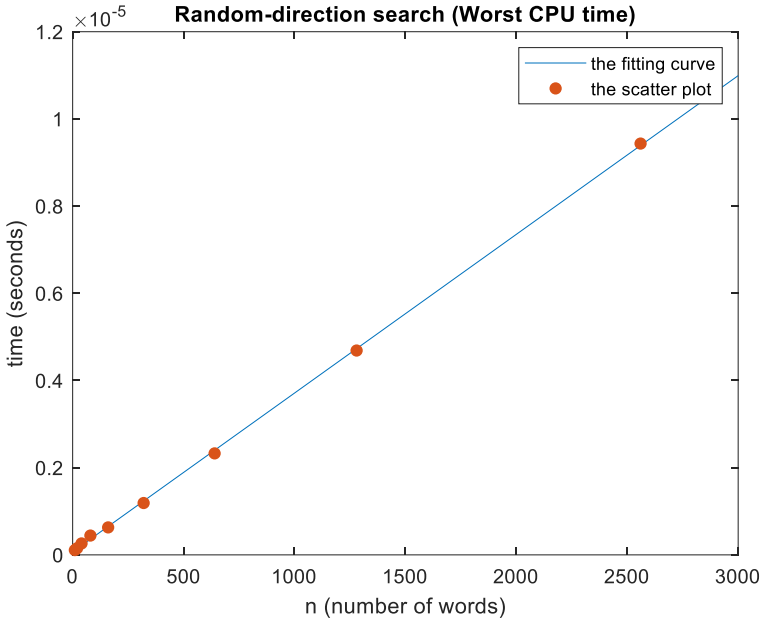

Figure 6 Random-direction search result (Worst). The result of the worst-case CPU time vs. n is linear. The result consists with the time complexity O(n) which I analyzed before.

The following figures are the comparison of three algorithms:

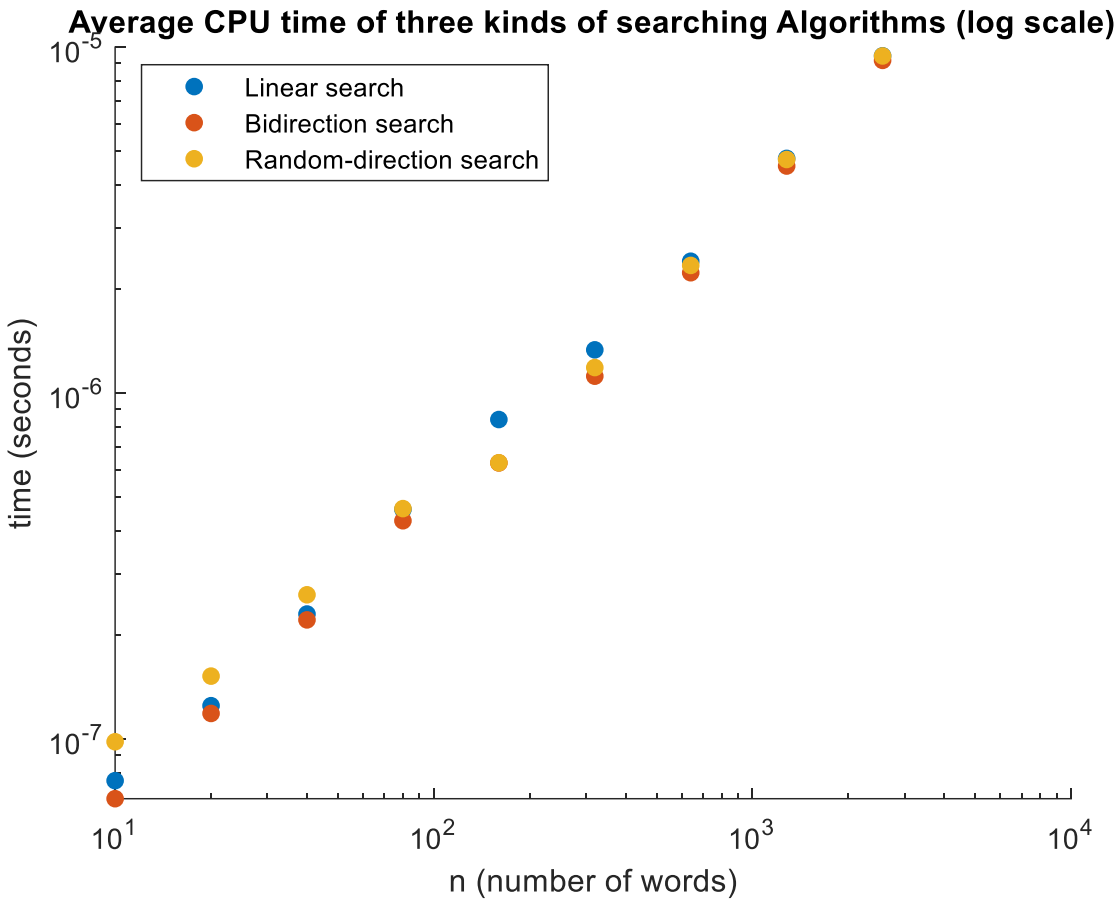

Figure 7 Average CPU time of three kinds of searching Algorithms.

According to **Figure 7,** the average CPU time of three searching Algorithms are pretty close when n is sufficiently. It consists with **Table 5** that the time complexities of Linear search, Bidirection search, and random-direction search are n+1, n+2, and n respectively.

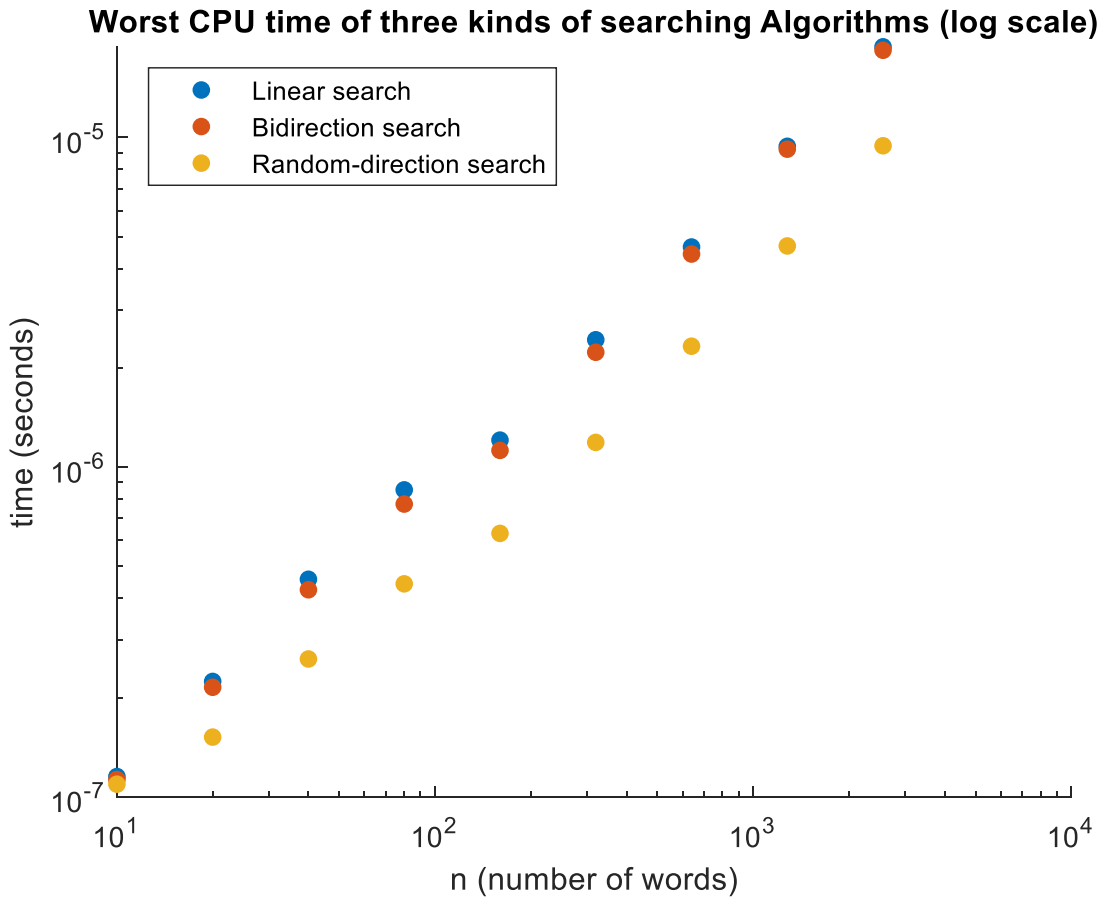

Figure 8 Worst CPU time of three kinds of searching Algorithms.

The worst-case time complexity of Linear search, Bidirection search, and Random-direction search are 2n, 3n / 2, n, respectively. Thus, Random-direction search is faster than Bidirection search, and Bidirection search is faster than Linear search. It consists with **Figure 8**.