**최악의 상태일 때 알고리즘의 분석과 그 결과**

알고리즘

숭실대학교 소프트웨어학부

20150213 김규호

신정훈 교수님

주어진 알고리즘들이 최악의 경우에 어떤 시간복잡도를 가지게 되는지를 살펴보고 각 상황에 알고리즘의 우수성을 고찰한다.

목차

1. 탐색

2. 피보나치

3. 정렬

4. 구현화면

5. 소스코트

1. 탐색

구현 전 : 순차 탐색과 이진 탐색을 할 때, 계산된 시간복잡도는 다음과 같다.

sequential search : O(n) binary search : O(log(n))

따라서 이진 탐색이 보다 더 빠른 시간 안에 주어진 것을 찾아 낼 것으로 예상된다.

구현 후 : x축 - 리스트의 길이(3000, 6000, 9000, 12000), y축 - 시간

그림 O(n). 그림 O(log(n))

둘 다 각 시간복잡도의 전형적인 모습을 보인다. 최선의 경우에는 조금 다른 양상을 보일 것으로 사료되나 이 테스트는 최악의 경우를 상정하였기에 n의 값(리스트의 크기)가 커질수록 이진 탐색의 탐색 시간이 더 효율적임을 알 수 있다.

|  |  |  |  |
| --- | --- | --- | --- |
| seq | | bin | |
| 3000 | 0.0093 | 3000 | 0.0000859 |
| 6000 | 0.0203 | 6000 | 0.0000938 |
| 9000 | 0.0282 | 9000 | 0.0001015 |
| 12000 | 0.0375 | 12000 | 0.0001016 |

2. 피보나치

구현 전 : 재귀 호출을 이용한 방법과 반복을 사용한 방법으로 나뉘어진다. 계산된 알고리즘의 시간복잡도는 각각

iterative : O(n) recursive : O(2^n)

이다. 따라서 반복을 사용한 방법이 매우 효율이 좋을 것으로 예상된다.

구현 후 : x 축 - 리스트의 길이(3000/10, 6000/20, 9000/30, 12000/40), y축 - 시간

그림 O(n) 그림 O(2^n)

그래프를 통해 직관적으로 유추해 낼 수 있듯이 반복을 사용한 방법이 매우 효율적임을 알 수 있다. 특히 재귀를 사용한 방법의 경우 시간복잡도 O(2^n)에 걸맞게 증가량이 폭발적으로 상승하고 있음을 볼 수 있다.

|  |  |  |  |
| --- | --- | --- | --- |
| fibite | | fibrec | |
| 3000 | 0.01062 | 10 | 0.000531 |
| 6000 | 0.02157 | 20 | 0.0661 |
| 9000 | 0.03187 | 30 | 8.28 |
| 12000 | 0.0425 | 40 | 1000 |

3. 정렬

구현 전 : 퀵 정렬과 합병 정렬의 경우 매우 빠른 정렬 알고리즘으로 알려져 있다. 특히 평균적인 시간복잡도는 각각 다음과 같다.

mergesort : O(nlog(n)) quicksort : O(nlog(n))

따라서 두 정렬 알고리즘의 효율은 유사할 것으로 예상되어진다.

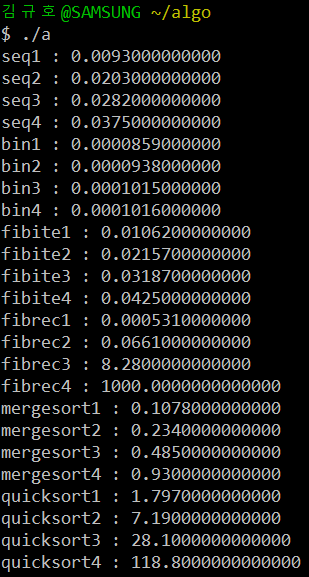
구현 후 : x축 - 리스트의 길이(1000, 2000, 4000, 8000), y축 - 시간

그림 O(nlog(n)) 그림 O(n^2)

합병 정렬의 경우 예상했던 대로 O(nlog(n))의 시간복잡도를 보여주었다. 반면에 비슷한 양상을 보일 것이라고 예상했던 퀵 정렬의 경우 O(n^2)이라는 시간복잡도를 보여주었다. 최악의 경우를 상정하였을 때 퀵 정렬의 효율이 떨어진다는 것을 확인할 수 있다.

|  |  |  |  |
| --- | --- | --- | --- |
| merge | | quick | |
| 3000 | 0.1078 | 3000 | 1.797 |
| 6000 | 0.234 | 6000 | 7.19 |
| 9000 | 0.485 | 9000 | 28.1 |
| 12000 | 0.93 | 12000 | 118.8 |

4. 구현화면



5. 소스코드

#include<cstdio>

#include<iostream>

void seqsearch(int n, const int S[], int x, int& loc)

{

loc = 0;

while(loc < n && S[loc] != x)

loc++;

if(loc >= n)

loc = 0;

}

void binsearch(int n, const int S[], int x, int& loc)

{

int low, high, mid;

low = 0;

high = n - 1;

loc = n;

while(low <= high && loc == n)

{

mid = (low + high)/2;

if(x == S[mid])

loc = mid;

else if(x < S[mid])

high = mid - 1;

else

low = mid + 1;

}

}

int fibrec(int n)

{

if(n <= 1)

return n;

else

return fibrec(n - 1) + fibrec(n - 2);

}

int fibite(int n)

{

int i;

int f[n];

f[0] = 0;

if(n > 0)

{

f[1] = 1;

for(i = 2; i <= n; i++)

f[i] = f[i - 1] + f[i - 2];

}

return f[n];

}

void merge(int S[], int low, int mid, int high)

{

int U[10000];

int i = low, j = mid + 1, k = 0;

while (i <= mid && j <= high) {

if (S[i] <= S[j])

U[k++] = S[i++];

else

U[k++] = S[j++];

}

while (i <= mid)

U[k++] = S[i++];

while (j <= high)

U[k++] = S[j++];

for(int i = 0; i < k; i++)

S[low + i] = U[i];

}

void mergesort(int S[], int low, int high)

{

if (low < high) {

int m = (high + low)/2;

mergesort(S, low, m);

mergesort(S, m + 1, high);

merge(S, low, m, high);

}

}

void partition(int S[], int low, int high, int& piv)

{

int i, j;

int temp;

int pivot;

pivot = S[low];

j = low;

for(i = low + 1; i <= high; i++)

if(S[i] < pivot)

{

j++;

temp = S[i];

S[i] = S[j];

S[j] = temp;

}

piv = j;

temp = S[low];

S[low] = S[piv];

S[piv] = temp;

}

void quicksort(int S[], int low, int high)

{

int piv;

if(high > low)

{

partition(S, low, high, piv);

quicksort(S, low, piv - 1);

quicksort(S, piv + 1, high);

}

}

int main(void)

{

double start, end;

double result;

int loc;

int A1[12000], A2[9000], A3[6000], A4[3000];

for(int i = 0; i < 12000; i++)

{

A1[i] = 12000 - i;

}

for(int i = 0; i < 9000; i++)

{

A2[i] = 9000 - i;

}

for(int i = 0; i < 6000; i++)

{

A3[i] = 6000 - i;

}

for(int i = 0; i < 3000; i++)

{

A4[i] = 3000 - i;

}

start = clock();

for(int i = 0; i < 10000; i++)

seqsearch(3000, A1, 0, loc);

end = clock();

result = (end - start) / 10000;

printf("seq1 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 10000; i++)

seqsearch(6000, A1, 0, loc);

end = clock();

result = (end - start) / 10000;

printf("seq2 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 10000; i++)

seqsearch(9000, A1, 0, loc);

end = clock();

result = (end - start) / 10000;

printf("seq3 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 10000; i++)

seqsearch(12000, A1, 0, loc);

end = clock();

result = (end - start) / 10000;

printf("seq4 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 10000000; i++)

binsearch(3000, A1, 0, loc);

end = clock();

result = (end - start) / 10000000;

printf("bin1 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 10000000; i++)

binsearch(6000, A1, 0, loc);

end = clock();

result = (end - start) / 10000000;

printf("bin2 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 10000000; i++)

binsearch(9000, A1, 0, loc);

end = clock();

result = (end - start) / 10000000;

printf("bin3 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 10000000; i++)

binsearch(12000, A1, 0, loc);

end = clock();

result = (end - start) / 10000000;

printf("bin4 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 100000; i++)

fibite(3000);

end = clock();

result = (end - start) / 100000;

printf("fibite1 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 100000; i++)

fibite(6000);

end = clock();

result = (end - start) / 100000;

printf("fibite2 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 100000; i++)

fibite(9000);

end = clock();

result = (end - start) / 100000;

printf("fibite3 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 100000; i++)

fibite(12000);

end = clock();

result = (end - start) / 100000;

printf("fibite4 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 1000000; i++)

fibrec(10);

end = clock();

result = (end - start) / 1000000;

printf("fibrec1 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 100000; i++)

fibrec(20);

end = clock();

result = (end - start) / 100000;

printf("fibrec2 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 100; i++)

fibrec(30);

end = clock();

result = (end - start) / 100;

printf("fibrec3 : %.13lf\n", result);

start = clock();

fibrec(40);

end = clock();

result = end - start;

printf("fibrec4 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 10000; i++)

mergesort(A4, 0, 1000);

end = clock();

result = (end - start) / 10000;

printf("mergesort1 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 1000; i++)

mergesort(A3, 0, 2000);

end = clock();

result = (end - start) / 1000;

printf("mergesort2 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 1000; i++)

mergesort(A2, 0, 4000);

end = clock();

result = (end - start) / 1000;

printf("mergesort3 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 100; i++)

mergesort(A1, 0, 8000);

end = clock();

result = (end - start) / 100;

printf("mergesort4 : %.13lf\n", result);

for(int i = 0; i < 12000; i++)

{

A1[i] = 12000 - i;

}

for(int i = 0; i < 9000; i++)

{

A2[i] = 9000 - i;

}

for(int i = 0; i < 6000; i++)

{

A3[i] = 6000 - i;

}

for(int i = 0; i < 3000; i++)

{

A4[i] = 3000 - i;

}

start = clock();

for(int i = 0; i < 1000; i++)

quicksort(A4, 0, 1000);

end = clock();

result = (end - start) / 1000;

printf("quicksort1 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 100; i++)

quicksort(A3, 0, 2000);

end = clock();

result = (end - start) / 100;

printf("quicksort2 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 10; i++)

quicksort(A2, 0, 4000);

end = clock();

result = (end - start) / 10;

printf("quicksort3 : %.13lf\n", result);

start = clock();

for(int i = 0; i < 10; i++)

quicksort(A1, 0, 8000);

end = clock();

result = (end - start) / 10;

printf("quicksort4 : %.13lf\n", result);

return 0;

}