**2017-2 알고리즘분석 과제**

**2016104114 김 우재**

1. **관련 이론 소개**

* 순차 탐색 (Sequential Search)

데이터 배열을 처음부터 끝까지 차례대로 배열하여 원하는 데이터를 찾아내는 알고리즘이다. 데이터를 따로 조작할 필요가 없어 단순하지만 비효율적이라는 단점을 가지고 있다. 단 방향 탐색을 수행하기 때문에 선형 탐색(Linear Search)이라고 부르기도 한다. O(N)의 시간복잡도를 가진다.

* 이분 탐색 (Binary Search)

데이터 배열을 두 부분으로 분할하여 원하는 데이터를 찾아내는 알고리즘이다. 데이터 배열 전부를 탐색하는 순차 탐색 기법에 비하여 빠르다는 장점이 있다. 하지만, 데이터가 정렬되어 있어야 (Sorted) 하고, 순차 탐색 기법에 비해 알고리즘 구조가 복잡하다는 단점이 있다. O(log N)의 시간복잡도를 가진다.

1. **시간 측정 프로그램 설계**
2. 데이터 설정

SIZE를 define 한 후, SIZE에 맞게 integer형 배열을 전역으로 설정한다.

1. 시간 측정 함수 설계

clock() 함수를 이용하여 시간 값을 밀리 세컨드(ms) 단위로 받아온 뒤, CLOCKS\_PER\_SEC 로 나누어 초 단위로 수행 시간을 구한다.

현재 시간을 받아오는 cpu\_time() 함수를 만들어 탐색 알고리즘 수행 전후로 시행하여 수행 시간을 구한다.

1. 탐색 알고리즘 설계

순차 탐색 알고리즘과 이분 탐색 알고리즘을 설계한다. 수행 시간을 측정하는 것이 목표이므로 원하는 값을 찾을 경우 index 값을 반환하기 보다는 함수를 끝내는 방법을 택하였다.

1. 탐색 알고리즘 반복 설계

순차 탐색 알고리즘과 이분 탐색 알고리즘 각각에 대하여 모든 값을 탐색하는 알고리즘을 설계하여 그 함수 이름을 각각 search1(순차 탐색), search2(이분 탐색) 이라 한다.

1. SIZE값 (=n) 설정

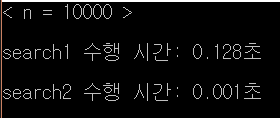
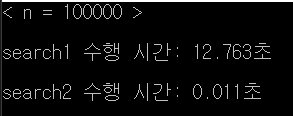
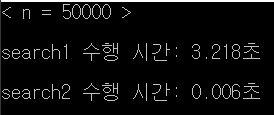
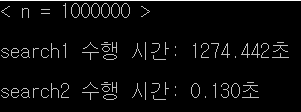
배열의 크기를 임의의 정수 값으로 3개를 정한다.

n1 = 10000, n2 = 50000, n3 = 100000 로 정하였다. 추가적으로 n = 1000000 을 정하였다.

1. **프로그램 실행 결과**

프로그램 실행 결과를 아래에 콘솔 스크린과 그 결과를 이용한 그래프로 나타내었다.

1. Console Screen Capture

1. 그래프
2. **수행 결과 예측**

위 측정 결과를 바탕으로 n = 50000000 일 때의 수행 시간을 추정 해 보았다.

1. Search1 (순차 탐색)

n값이 약 10배 증가함에 따라 수행 시간이 약 100배 증가하는 것을 분석할 수 있었다. 따라서 1000000의 약 50배인 50000000에 대하여 순차 탐색을 시행할 경우, 수행 시간은 약 2500배 증가할 것으로 예측할 수 있다. 따라서 수행시간 t에 대하여

t = 1274.442 \* 2500 = 3186105 초 = 약 885 시간 이 걸린다고 짐작할 수 있다.

1. Search2 (이분 탐색)

n값이 약 10배 증가함에 따라 수행 시간이 약 10배 증가하는 것으로 분석할 수 있었다. 실제로, O(N log N)의 시간복잡도를 가진 함수이므로 약 10 log 10배 증가한다고 할 수 있을 것 이다. n = 50000000에 대하여 실제로 수행 하니 8.226 초 의 결과를 얻을 수 있었다.

1. **프로그램 코드**

수행한 프로그램의 코드 전문을 아래에 첨부한다.

#include <iostream>

#include <ctime>

using namespace std;

#define SIZE 50000000

int arr[SIZE];

double cpu\_time() {

double value;

value = clock() / (double)CLOCKS\_PER\_SEC;

return value;

} // 시간 측정 함수

void find\_sequential(int n) {

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

if (arr[i] == n)

return;

}

} // 순차 탐색 알고리즘

void find\_binary(int n) {

int first = 0, last = SIZE - 1;

while (1) {

int mid = (first + last) / 2;

if (arr[mid] == n)

return;

else if (arr[mid] > n)

last = mid - 1;

else

first = mid + 1;

}

} // 이분 탐색 알고리즘

void search1() {

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

find\_sequential(i);

}

} // 모든 n값에 대하여 순차 탐색

void search2() {

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

find\_binary(i);

}

} // 모든 n값에 대하여 이분 탐색

int main() {

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

arr[i] = i;

} // 배열에 원소 대입

double before\_time, after\_time;

// 측정 시간을 저장할 변수 선언

cout << "< n = " << SIZE << " >" << endl << endl;

before\_time = cpu\_time();

search1(); // 순차 탐색 수행

after\_time = cpu\_time();

cout.precision(3);

cout << fixed << "search1 수행 시간: " << after\_time - before\_time << "초" << endl;

before\_time = cpu\_time();

search2(); // 이분 탐색 수행

after\_time = cpu\_time();

cout << fixed << "search2 수행 시간: " << after\_time - before\_time << "초" << endl;

}