알고리즘 설계와 분석 프로그래밍 숙제 1

2반

20141494

강동욱

A. Maximum Subsequence Sum Problem

1. 수행 환경

OS: Window 10 Education

CPU: AMD Ryzen 5 1600 3.70GHz

RAM: 16.00GB

Compiler: Visual Studio Community 17 (v141)

2. 실험 개요

사용한 데이터는 4바이트 int값에 저장되는 -100 ~ 100 사이의 값 n개의 집합이며, n의 크기를 로 증가시키면서 알고리즘과 수행시간 사이의 관계를 관찰한다.

각 n에 대한 수행시간은 서로 다른 5개의 데이터에 대해 실행한 값 중 중앙값을 선정하여 평가한다.

3.실험 결과

|  |  |  |  |
| --- | --- | --- | --- |
|  | A1 O() | A2 O() | A3 O(n) |
|  | 0.0000011 | 0.0000130 | 0.0000005 |
|  | 0.0000867 | 0.0001616 | 0.0000014 |
|  | 0.0216861 | 0.0025446 | 0.0000139 |
|  | 5.4500281 | 0.0416703 | 0.0001802 |
|  | 1416.59700 | 0.6693218 | 0.0029264 |
|  |  | 10.7129149 | 0.0473035 |
|  |  | 174.5817 | 0.738519 |

위 표에 대해 수행시간의 증가 폭에 대한 표를 작성해보면

|  |  |  |  |
| --- | --- | --- | --- |
|  | A1 O() | A2 O() | A3 O(n) |
|  | 78.81818 | 12.43077 | 2.8 |
|  | 250.128 | 15.74629 | 9.928571 |
|  | 251.3143 | 16.37597 | 12.96403 |
|  | 259.9247 | 16.06232 | 16.23973 |
|  |  | 16.00563 | 16.1644 |
|  |  | 16.29638 | 15.61235 |

와 같다.

사이즈 n이 작을 경우에는 상수시간이 걸리는 작업이 전체 소요시간에서 차지하는 비중이 크기 때문에 의미가 적다.

사이즈 n이 충분히 커졌을 때, 알고리즘 A1은 n이 16배 증가했을 때 n의 제곱인 256배에 근접하게 증가하는 모습을 보여 T(n)이고 O()임을 확인 할 수 있다.

알고리즘 A2,A3의 사이즈 n이 충분히 커졌을 때, 사이즈 n이 16배 커졌을 때 A3는 대략 16배, A2는 약 16.15배 증가하는 것을 확인 할 수 있다. A3의 경우 T(n)이므로 O(n)임을 확인 할 수 있다.

A2의 경우 n의 사이즈가 충분히 커졌을 때 이론상 n이 16배 증가한다면

이므로 n이 이론 상 18.6배 증가하여야 한다. 이러한 오차가 발생하는 가장 큰 이유는, divide and conquer 과정을 함수의 recursive 호출로 구현하였고, 이러한 함수호출과 복귀 과정은 다른 계산과정에 비해 시간이 매우 오래 걸린다. 이는 n이 작아 이론상 차이가 거의 없을 때에도 A3알고리즘과 100배 이상 차이 나는 수행시간에서도 확인 할 수 있다.

지금의 수행시간 T(n)은 과 같이 나타낼 수 있는데 함수호출 과정의 횟수인 에 호출시간 상수를 곱한 가 보다 훨씬 크기 때문에 발생했다고 볼 수 있다. 이러한 문제는 사이즈 n이 훨씬 커지거나 (이상으로 커져야 할 것으로 보이기 때문에 일반적으로 수행이 불가능하다. ), recursive 함수호출이 아닌 iterative한 방식을 사용하거나 꼬리재귀로 최적화해서 구현한다면 확실히 볼 수 있을 것이다.

B. Inversion Counting Problem

1. 수행 환경

OS: Window 10 Education

CPU: AMD Ryzen 5 1600 3.70GHz

RAM: 16.00GB

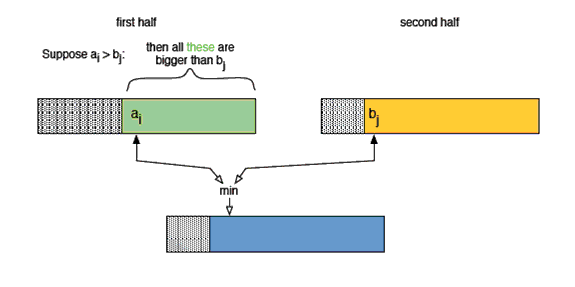
Compiler: Visual Studio Community 17 (v141)

2. 알고리즘 설계

Inversion이란, 어떤 수열 a에 대해서 .

어떤 수열에서 모든 inversion의 개수를 구하는 방법에 대해 생각해보면, 먼저 떠오르는 것은 수열의 모든 두 수의 쌍을 비교하여 inversion의 개수를 세는 방법을 생각 할 수 있다. 이 경우 시간복잡도는 O()이다.

하지만 merge sort와 같은 방식으로 정렬할 경우를 생각 해 보면,



( 그림 출처:http://www.geeksforgeeks.org/counting-inversions/ )

정렬된 두 수열을 merge 할 경우, 앞의 수열을 a 뒤의 수열을 b라고 하였을 때, 합치는 과정에서 a수열이 b수열의 앞부분에 있으므로, merge 과정에서 현재 비교하는 에 대하여 인 경우, 수열 a는 정렬되어 있으므로, 모든 수는 보다 크다. 따라서 두 수를 비교하는 과정에서 에 대한 inversion의 개수가 len(a) – i+1개 존재함을 알 수 있다. 이러한 merge과정은 시간복잡도 가 O(n) 이고, 이 과정을 각각 절반으로 나눠진 수열에 대해 재귀적으로 시행하므로, 시간복잡도가 O(nlogn)이다. 따라서 O(nlogn)시간 안에 inversion의 총 개수를 구할 수 있다.

3. 구현 과정

기존의 merge sort의 두 수열을 합치는 과정인 merge 과정에서 인 경우 inversion의 개수를 나타내는 변수 count에 middle-i+1을 더해준다. Count는 전역변수로 선언되어있다.

void merge\_sort(item\_type \*A, int left, int right) {

int middle;

if (left < right) {

middle = (left + right) / 2;

merge\_sort(A, left, middle);

merge\_sort(A, middle + 1, right);

merge(A, left, middle, right);

}

}

void merge(item\_type \*A, int left, int middle, int right) {

int i, i\_left, i\_right;

memcpy(buffer + left, A + left, sizeof(item\_type)\*(right - left + 1));

i\_left = left;

i\_right = middle + 1;

i = left;

while ((i\_left <= middle) && (i\_right <= right)) {

if (buffer[i\_left] < buffer[i\_right])

A[i++] = buffer[i\_left++];

else {

A[i++] = buffer[i\_right++];

count += middle - i\_left + 1;

}

}

while (i\_left <= middle)

A[i++] = buffer[i\_left++];

while (i\_right <= right)

A[i++] = buffer[i\_right++];

}

이런 방식으로 기존의 merge sort에 합 연산 한 줄과 count에 대한 선언만을 추가함으로써 inversion counting을 시행할 수 있다.