**CSE3081 알고리즘설계와분석 HW2 보고서**

컴퓨터공학과 2학년 20181662 이건영

1. **실험 환경**

**Operation System** : Microsoft Windows 10 Home (64-bit)

**Compiler** : Microsoft Visual Studio Community 2019

**CPU** : Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.80 GHz

**RAM** : 8.00GB

1. **코드 설명**
2. cmp 함수

텍스트이(가) 표시된 사진

자동 생성된 설명

전달받은 두 인자의 대소를 비교하여 같으면 0, a가 b보다 작으면 -1, 크면 1을 반환한다.

1. init\_array 함수

텍스트이(가) 표시된 사진

자동 생성된 설명ELEMENT 타입으로 이루어진 배열을 초기화하고 Fisher-Yates 알고리즘을 사용하여 값을 임의로 설정한다.

1. qsort\_orig 함수

텍스트, 모니터, 화면, 은색이(가) 표시된 사진

자동 생성된 설명

기본적으로 제공받은 qsort\_example 함수와 비슷하게 진행되나, pivot 설정 시 배열의 가장 왼쪽 원소를 사용하기 위해 가장 왼쪽에 있던 원소를 가장 오른쪽의 값과 바꿔 주는 과정을 거쳤다. 이를 통해 가장 왼쪽의 원소를 pivot으로 사용한 것과 같은 결과를 낼 수 있다.

텍스트, 화면, 은색, 스크린샷이(가) 표시된 사진

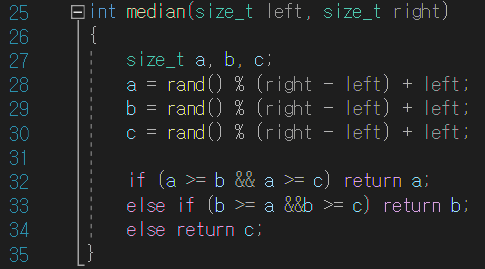
자동 생성된 설명

이 후의 과정 역시 qsort\_example과 비슷하게 진행된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. median 함수



pivot 설정을 위한 median 함수이다. 전달받은 값 사이에서 임의의 세 정수를 생성해 그 중앙값을 반환한다.

1. insertion\_sort 함수

텍스트, 모니터, 은색이(가) 표시된 사진

자동 생성된 설명

qsort\_median\_insert 함수의 남은 input size가 임계값보다 작아졌을 경우 실행하는 삽입 정렬 함수이다.

1. qsort\_median\_insert 함수

텍스트이(가) 표시된 사진

자동 생성된 설명

우선 남은 input size가 임계값 이하인지 확인하는 조건문으로 시작된다. 만약 조건에 해당할 경우 남은 부분에 대해서는 insertion sort 함수가 실행된다.

그렇지 않은 경우 quick sort 부분으로 넘어간다. 위의 qsort\_orig 함수와 다른 점은 median 함수를 이용하여 임의의 pivot 값을 정한다는 것이다. 이 역시 pivot 값의 위치를 정한 후 해당 위치의 값과 가장 오른쪽 값을 바꾸어 정렬을 시작한다. 이를 통해 median 함수로 선택된 값을 pivot으로 사용하는 것과 동일한 효과를 얻는다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

이하 부분은 qsort\_orig와 동일하나 재귀적으로 불러주는 함수가 qsort\_median\_insert 라는 점만 다르다.

1. qsort\_median\_insert\_iter 함수



추후 함수에서 사용되는 flag와 oldn 변수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

우선 flag값을 확인한다. 함수의 첫 번째 동작은 qsort가 이루어져야 하기 때문에 한 번의 동작 이후 flag값을 1로 만든다. 이후 이전 동작의 n 값의 절반에 해당하는 oldn / 2와 현재의 n값을 비교해 더 큰 부분일 경우 insertion\_sort를 통한 비재귀 방식의 정렬을 시행하도록 하였다.

그 이외의 경우인 quick\_sort 부분의 경우 임계값의 존재를 제외한다면 qsort\_median\_insert 함수와 동일하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

qsort\_median\_insert 함수와 동일하다.

텍스트, 전자기기이(가) 표시된 사진

자동 생성된 설명

qsort\_orig와 동일하나 재귀적으로 호출하는 함수가 qsort\_median\_insert\_iter 라는 점이 다르며, while loof 밖에 flag를 1로 설정하는 부분과 현재의 n값을 oldn에 저장하여 다음 함수 실행 시 사용될 수 있도록 한다.

1. **실행 결과**

각 함수(1번, 21번, 22번, 23번)에 대해서 5가지의 input size(1024, 2048, 4096, 32768, 131072)를 준비하였고, 각각 5회씩 실행한 결과의 평균을 구했다.

1. 1번 함수(stdlib.h의 qsort)
2. input size : 1024





텍스트, 시계, 게이지이(가) 표시된 사진

자동 생성된 설명



텍스트, 시계, 게이지이(가) 표시된 사진

자동 생성된 설명

평균 : 1.547ms

1. input size : 2048











평균 : 3.4682ms

1. input size : 4096



텍스트, 시계이(가) 표시된 사진

자동 생성된 설명



텍스트, 시계이(가) 표시된 사진

자동 생성된 설명



평균 : 8.7378ms

1. input size : 32768











평균 : 97.6784ms

1. input size : 131072











평균 : 333.2322ms

1. 21번 함수(qsort\_orig)
2. input size : 1024

텍스트이(가) 표시된 사진

자동 생성된 설명

평균 : 1.647ms

1. input size : 2048

텍스트이(가) 표시된 사진

자동 생성된 설명

평균 : 3.471ms

1. input size : 4096

텍스트이(가) 표시된 사진

자동 생성된 설명

평균 : 7.671ms

1. input size : 32768

텍스트이(가) 표시된 사진

자동 생성된 설명

평균 : 89.323ms

1. input size : 131072

텍스트이(가) 표시된 사진

자동 생성된 설명

평균 : 271.397ms

1. 22번 함수(qsort\_median\_insert)
2. input size : 1024

텍스트이(가) 표시된 사진

자동 생성된 설명

평균 : 1.123ms

1. input size : 2048

텍스트이(가) 표시된 사진

자동 생성된 설명

평균 : 2.537ms

1. input size : 4096

텍스트이(가) 표시된 사진

자동 생성된 설명

평균 : 7.610ms

1. input size : 32769

텍스트이(가) 표시된 사진

자동 생성된 설명

평균 : 58.674ms

1. input size : 131072

텍스트이(가) 표시된 사진

자동 생성된 설명

평균 : 244.031ms

1. 23번 함수(qsort\_median\_insert\_iter)
2. input size : 1024

텍스트이(가) 표시된 사진

자동 생성된 설명

평균 : 4.116ms

1. input size : 2048

텍스트이(가) 표시된 사진

자동 생성된 설명

평균 : 5.580ms

1. input size : 4096

텍스트이(가) 표시된 사진

자동 생성된 설명

평균 : 26.543ms

1. input size : 32768

텍스트, 화살이(가) 표시된 사진

자동 생성된 설명

평균 : 1832.753ms

1. input size : 131072

텍스트, 화살이(가) 표시된 사진

자동 생성된 설명

평균 : 19018.771ms

1. **결과 분석 및 제언**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| (단위 : ms) | 함수 1번 | 함수 21번 | 함수 22번 | 함수 23번 |
| 1024 | 1.547 | 1.647 | 1.123 | 4.116 |
| 2048 | 3.4682 | 3.471 | 2.537 | 5.580 |
| 4096 | 8.7378 | 7.671 | 7.610 | 26.543 |
| 32768 | 97.6784 | 89.323 | 58.674 | 1832.753 |
| 131072 | 333.2322 | 271.397 | 244.031 | 19018.771 |

본 프로그램의 함수 별 입력 크기에 따른 실행 시간을 표로 정리해보았다.

가장 먼저 눈에 띄는 것은 23번 함수가 1번~22번 함수에 비해서 확연하게 느린 성능을 보여주고 있다는 것이다. 그 원인은 비교적 시간복잡도가 높은 삽입정렬(Avg case : O(n^2))을 크기가 큰 부분에 대해서 실행하였기 때문이라고 생각한다.

다음으로 1번 함수와 21번 함수는 그 실행 속도에서 큰 차이를 보이지 않았다. 오히려 큰 input size에 대해서 근소하게 빠른 모습을 보였는데, 이를 통해 pivot을 왼쪽 끝에서 선정한 후 위치를 오른쪽 끝으로 바꾸는 동작이 실행 속도에는 큰 영향을 주지 않았음을 알 수 있었다.

그러나 22번 함수와 1번 함수는 큰 입력 크기에 대해서 입력 속도에서 유의미한 차이를 보였다. 이는 작은 n에 대한 insertion sort 선택과 단순히 왼쪽 원소를 pivot으로 택하는 것이 아닌 무작위적으로 pivot을 선정함으로써 worst case를 피할 확률이 높아졌기 때문이라고 볼 수 있다.

또한 22번 함수와 23번 함수를 비교하였을 때 22번 함수는 큰 부분에 대해서 재귀를 시행한 후 입력 크기가 충분히 작아졌을 때 반복형 함수를 동작한 반면 23번 함수는 반대로 큰 부분에 대해서는 반복으로, 작은 부분에는 재귀로 접근하였다. 그 결과 22번 함수에 비해 23번 함수의 실행 속도가 느린 것으로 나타났다. 이를 통해 작은 input size에 대해 재귀를 실행하는 것이 무조건적인 성능 향상을 가져다 주는 것이 아님을 알 수 있었다.