알고리즘 과제2

소프트웨어학과 32203660 이현정

1. 개발 환경 : Ubuntu 환경에서 해당 프로그램을 만들어보았습니다.

```
hyeonjeong@LAPTOP-I49JJJGR:/mnt/c/Users/1212g/OneDrive - 단국대학교/바탕 화면/대학/대학 22-1/알고리즘/wor
k2.c$ gcc -o sort sort.c
```

```
hyeonjeong@LAPTOP-I49JJJGR:/mnt/c/Users/1212g/OneDrive - 단국대학교/바탕 화면/대학/대학 22-1/알고리즘/wor
k2.c$ ./sort
exchange sort 100 times : 28
merge sort 100 times : 10
quick sort 100 times : 6
heap sort 100 times : 42
```

2. 프로그램 설명

2-1) exchange 정렬

가장 기본적인 정렬로 2중 for문을 돌면서 2개씩 값을 비교해서 앞의 값이 뒤의 값보다 큰 경우에는 2값에 대한 자리 변동이 발생하게 되고, 해당하는 내용이 바로 전달받은 배열에서 발생되게 되는 구조를 가집니다.

2-2) merge 정렬

mergesort() : 시작 인덱스와 끝 인덱스+1의 값을 low와 high로 받은 후에 주어지는 범위가 1보다 큰 경우에는 중간 값을 mid에 담고 앞쪽 배열과 뒤쪽 배열로 mid값을 기준으로 해당 범위를 나눈 후에 해당 값의 시작과 끝+1 인덱스 값들을 배열과 함께 넘겨줘서 각 정렬될 배열의 범위를 알려주면서 해당 범위가 분리되게 구성하였습니다.

merge() : low, mid(= 뒤쪽 배열의 시작 위치), high의 값을 받아서 2 배열의 요소들을 비교하면서 더 작은 것을 새로운 배열인 new에 담고, 2개 중 1개라도 다 담기는 경우에는 다 담기지 못한 값을 해당 범위에 속하는 만큼(= 해당 범위 값) 끝까지 new에 마저 담게 만들었습니다. 그리하여 만들어진 새로운 정렬을 주어진 원래 배열의 해당 범위에 담는 방식으로 각 범위가 정렬되게 됩니다.

2-3) quick 정렬

quicksort() : merge와 구조는 비슷하지만, 먼저 pivot의 위치를 정리하고 앞과 뒤의 배열들에 대해서 다시 정렬을 위해 pivot를 이용하는 구조를 가집니다. 현재 정렬할 배열의 가장 작은 값과 가장 큰 값보다 1개 큰 값을 받아서 low~high 사이의 값에 대해서 pivot를 찾게 됩니다.

partition() : pivot을 정리하는 부분에 대한 함수로 이 부분을 이용해서 결국 모든 값들이 pivot이 되어 자신의 자리를 찾아서 정렬이 되게 됩니다. pivot의 값보다 작은 경우를 가지고 작은 값과 위치가 변경될 인덱스 j를 low+1~high-1까지 비교하면서 작은 값의 개수만큼 값을 키웁니다. 그래서 변경이 다 되었을 때, j는 마지막 변경한 pivot보다 작은 값의 위치를 가지고 있으며 해당 위치와 pivot을 삼았던 값의 위치를 바꾸면 pivot이 자신의 자리를 찾고 나머지 값들이 위치에 따라 pivot보다 작은 값과 큰 값으로 나누어집니다.

2-4) heap 정렬

createheap() : 기본적으로 구조체를 타입처럼 이용할 수 있는 heaptype을 만들고, 힙을 h라는 이름으로 만들어서 heap에 대한 초기화를 진행하였습니다.

insertheap() : 들어오는 값이 정렬 없이 바로 들어오는 대로 힙에 들어가게 하였습니다.

heapsort() : makeheap()과 removeheap()를 통해서 주어진 배열의 값이 정렬되게 됩니다.

makeheap() : 정렬되지 않게 넣은 값들에 대해서 left의 부모부터 천천히 올라오면서 자식의 값들이 siftdown에 의해서 처리되게 합니다.

removeheap() : root을 값을 가져와서 해당 값을 결과를 저장할 result에 담아서 정렬 값을 확인할 수 있게 합니다.

root() : 배열의 인덱스1에 가장 큰 값이 해당하므로 그 값을 넘겨주기 위해서 전체적인 크기를 조정과 가장 아래의 값을 root에 넣었기 때문에 정리를 위해서 siftdown()을 호출해서 이용합니다.

siftdown() : 받은 인덱스 값을 기준으로 오른쪽이 있으면서 오른쪽 값이 왼쪽보다 큰 경우에는 오른쪽에 넣어져야한다고 둘 중 큰 값을 가질 b_child을 par*2+1로 만들고, 왼쪽 값이 큰 경우에는 par*2로 만들게 됩니다. 그리고 자신이 b_child보다 작은 경우(= 자식 2개보다 자신이 작음)에는 자신을 나타내던 i를 b_child쪽으로 보내고 위에서 한 작업을 반복합니다. 그렇지 않으면, 자신이 2개보다 더 큰 것으로 해당 자리를 만족하므로 해당 자리에 자신의 값을 넣게 됩니다. 이러한 과정을 통해서 각 주어진 인덱스는 맞는 자리에 위치하게 됩니다.

2-5) radix 정렬

radixsort() : 기본적으로 node 구조체를 사용해서 다음 값을 가리키고 자신의 값을 가집니다. distribute()를 통해서 자리의 값에 따라서 나눠지고, coalesce()에 따라서 나눠져서 처리된 값들이 합쳐지게 됩니다.

insertnode() : 값들을 노드로 삽입하기 위한 함수로, 시작하는 값일 때는 해당 포인터에 직접 값을 넣고, 이미 노드가 존재할 때는 마지막으로 이동해서 NULL인 경우에 노드를 만들어서 넣게 하였습니다.

distribute() : 이미 값이 담겨진 master를 돌면서 각 값들의 현재 가리키는 자릿수에 따라서 list의 각 위치의 마지막의 노드가 되어서 해당 노드들이 들어가게 됩니다.

coalesce() : list에 정리된 값들을 다시 1개의 링크드 리스트로 정리하기 위해서 각 값을 순서대로 꺼내서 연결합니다.

3. 실행 결과 및 분석

0~99를 랜덤으로 생성해서 각 함수마다 정해진 5개의 횟수를 5번씩 돌려서 평균적인 시간을 나타낸 것입니다.

```
exchange sort 100 times : 24
merge sort 100 times : 8
quick sort 100 times : 6
heap sort 100 times : 38

exchange sort 500 times : 490
merge sort 500 times : 51
quick sort 500 times : 34
heap sort 500 times : 252

exchange sort 1000 times : 1659
merge sort 1000 times : 145
quick sort 1000 times : 79
heap sort 1000 times : 692

exchange sort 5000 times : 27997
merge sort 5000 times : 634
quick sort 5000 times : 647
heap sort 5000 times : 2975

exchange sort 10000 times : 100258
merge sort 10000 times : 1199
quick sort 10000 times : 1719
heap sort 10000 times : 5985
```

merge나 quick 정렬이 평균적으로 가장 빠르며, 그 다음으로는 heap 정렬이 100개인 경우를 제외하고 빠르다는 것을 확인할 수 있습니다. exchange 정렬은 다른 정렬에 비해서 개수가 많아질수록 걸리는 시간이 매우 증가하는 경향을 보이고 있습니다.

exchange이 걸리는 시간의 절반 정도가 heap이 걸리고, heap의 1/5정도로 merge와 quick정렬을 걸립니다.

4) Github 링크

https://github.com/12hyeon/Hyeon-Algorithm/blob/main/lec_algorithm/sort.c