ACE+
잘 가르치는 대학

DU-도전학기 결과보고서

성 명		학 번	
단과대학		학과(전공)	
도전학기 과제명	직접 만들며 배우는 정렬 알고리즘 Make and Learn myself about Sort Algorithms		
지도교수 의견	최종 결과물을 통하여 3가지 언어로 6가지 정렬 알고리즘 구현 내용을 확인하고, 이를 통해 수업시간에 배운 동일한 문제를 해결하는데 다양한 방법을 비교해보고 다른 학생들에게도 도움이 되는 (설명) 유익한 연구 성과를 도출함. (설명)		

1. 도전 과제의 목표

3가지의 고급정렬 알고리즘을 서로 다른 3개의 프로그래밍 언어(C언어, Java, Python)로 제작하여 학습하며, 알고리즘별 특성을 확인하고 GitHub 사이트에 코드들과 언어간 특성을 공유한다.

최종적으로는 도전 학기 간 제작하며 학습한 6개의 알고리즘을 설명할 수 있다.

2. 도전 과제 내용

지난 3주간 진행했을 당시에는 언어별로 정렬 알고리즘의 차이에 집중하여 확인했는데, 지도 교수님과의 중간 면담에서 언어별 소요시간에 대해서는 사람이 큰 차이를 느끼기 힘들 것 같으므로 정렬별 비교 횟수와 교환 횟수를 세어 알고리즘 간 횟수의 차이를 확인하는 것이 더 좋을 것 같다는 의견을 주셔서, 기존에 작성된 코드들을 모두 수정한 뒤 GitHub 사이트에도 수정된 코드를 등록했다.

선택정렬	비교횟수	교환횟수	평균적으로 (n (n-1)) / 2에 근접한 횟수의 비교를 진행하고, 원소의 위치에 상관없이 배열 전체를 훑고 난 뒤에 교환이 일어나기 때문에 교환 횟수가 배열의 크기와 동일하게 나왔다.
	499500	999	
버블정렬	비교횟수	교환횟수	평균적으로 (n (n-1)) / 2에 근접한 횟수의 비교를 진행하고, 교환 횟수는 대체적으로 (n (n-1)) / 4에 근접한 횟수임을 볼 수 있었다.
	498802.7	251810.6	
삽입정렬	비교횟수	교환횟수	평균적으로 (n (n-1)) / 2보다 더 적은 수의 비교가 진행되고, 원소가 새 자리에 들어갈 때 교환 횟수가 증가하도록 하여 교환 횟수가 배열의 크기와 동일하게 나왔다.
	24581.2	999	

(1000개의 원소 배열에 대해 언어별 5회 총 15회 비교, 교환 횟수의 평균값을 낸 자료

이다)
다음으로 4~6주간 진행한 고급정렬 알고리즘 학습에 대한 내용이다.

고급정렬 알고리즘으로는 병합정렬, 퀵 정렬, 힙 정렬이 있으며, 4주 차에는 병합정렬을, 5주 차에는 퀵 정렬을, 6주 차에 힙 정렬 코드를 제작했다.

4주차: 병합정렬 알고리즘 코드를 언어별로 제작하였다. 병합정렬은 전체원소를 하나의 단위로 분할한 뒤에 분할된 원소들을 다시 합치면서 정렬이 된다. 정렬을 하게 되면 중간값을 기준으로 양 옆의 두 구간이 생기는데, 이 두 구간에 대해서도 다시 병합정렬이 재귀적으로 호출된다. 예를 들어, 8개의 서로 다른 값을 가지는 배열이 있다.

6	5	3	1	8	7	2	4

이 배열이 분할되며 중간을 기준으로 양쪽 구간에 대해 병합정렬을 다시 진행한다.

6	5	3	1	8	7	2	4

두 개로 나뉜 병합정렬은 다시 각각의 병합정렬에서 중간을 기준으로 분할시킨다.

6	5	3	1	8	7	2	4

아직 하나의 단위로 나뉘지 않았으므로 한 번 더 분할이 되고나면,

6	5	3	1	8	7	2	4

드디어 하나의 단위로 나뉘게 되었다. 이후에 오름차순으로 정렬을 한다고 가정하면

5	6	1	3	7	8	2	4

병합이 시작되면서 크기가 작은 순으로 원소들이 정렬되기 시작한다.

1	3	5	6	2	4	7	8

여기서 주목해야 할 점은 병합 시 단순히 그룹이 옮겨진다고 생각할 수도 있지만, 원소들이 제일 작은 수 1부터 2, 3...7, 8까지 크기순으로 하나씩 옮겨진다는 점이다.

1	2	3	4	5	6	7	8

이와 같이 하나의 단위까지 분할된 뒤에, 다시 원소들을 합치며 정렬시키는 정렬이 바로 병합정렬이다.

시간복잡도는 기본정렬 알고리즘들이 최악의 경우 $O(n^2)$의 시간복잡도를 가지는데 비

해 병합정렬은 최악의 경우에도 O(nlogn)의 시간복잡도만을 가진다.

5주차: 퀵 정렬 알고리즘 코드를 언어별로 제작하였다. 퀵 정렬은 평균적으로 성능이 좋아 가장 많이 쓰이는 정렬 알고리즘이며, 정렬 원리로는 기준 원소 하나를 선택하여 그 기준 원소보다 작은 그룹과 큰 그룹으로 나뉘어 작은 그룹은 왼쪽에, 큰 그룹은 오른쪽에 재배치된다. 그 이후 각 그룹을 다시 정렬하는 과정에서 병합정렬과 동일하게 재귀 호출이 일어나는데, 다른 점이 있다면 병합정렬은 가장 말단까지 분할되고 나서 다시 합쳐지며 정렬이 이루어지지만, 퀵 정렬은 분할하는 과정에 기준 원소를 기준으로 재배치되기 때문에 분할을 할 때 정렬이 동시에 진행된다고 볼 수 있다. 병합정렬에서 본 배열을 예시로 보며 확인해보겠다.

6	5	3	1	8	7	2	4

이와 같이 배열이 주어질 때 마지막 원소 4를 기준 원소로 삼고 재배치를 한다.

3	1	2	4	6	5	8	7

다음에는 작은 그룹에서는 2를 기준 원소로, 큰 그룹에서는 7을 기준으로 재배치한다.

1	2	3	4	6	5	7	8

아직 큰 그룹에 정렬할 원소가 남아있으므로 5가 기준 원소로 재배치 되면

1	2	3	4	5	6	7	8

이와 같이 정렬된 모습을 볼 수 있고, 모든 재배치를 마치게 되면 정렬이 끝나게 된다.

1	2	3	4	5	6	7	8

시간복잡도는 최선의 경우와 평균적인 경우에 O(nlogn)의 시간 복잡도를 가지지만 이미 정렬이 되어 있는 자료에 대해 정렬을 하는 최악의 상황에 대해서는 $O(n^2)$의 시간 복잡도를 가지게 된다.

6주차: 힙 정렬 알고리즘 코드를 언어별로 제작하였다. 힙은 완전 이진 트리로 이루어지지만, 이를 배열로 구현하여 정렬을 할 수 있는데 이때는 힙의 구조 조건을 만족할 필요 없이 순서조건만 만족하면 된다. 간단한 예시를 통해 확인하자면

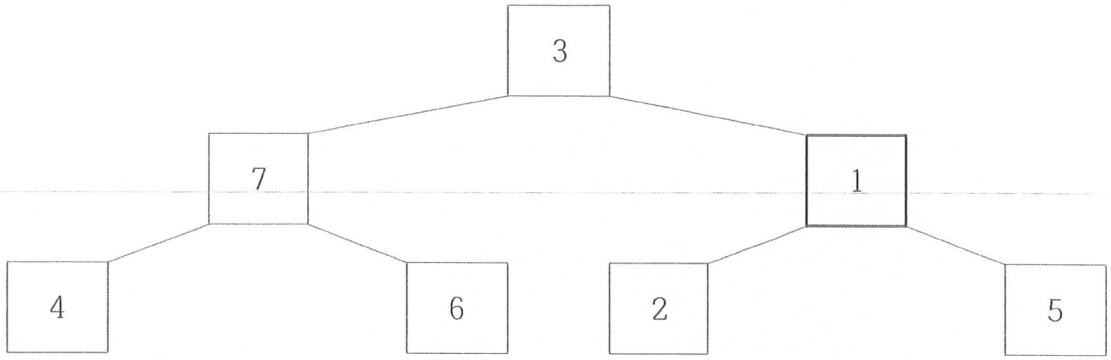
루트노드	5	3	6	2	1	7	4
0	1	2	3	4	5	6	7

이처럼 배열이 주어질 때 1번 인덱스의 자식은 1x2,. (1x2)+1 = 2번, 3번 인덱스이며, 2

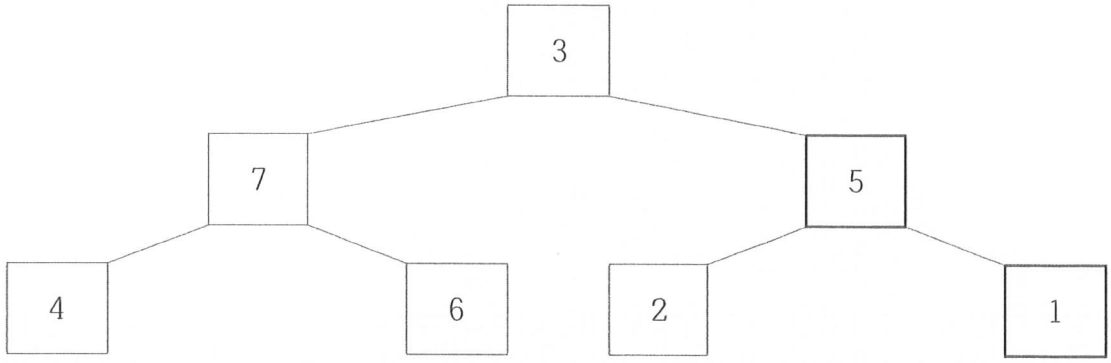
번 인덱스의 자식은 2x2, (2x2)+1 = 4번, 5번 인덱스, 3번 인덱스의 자식은 3x2, (3x2)+1 = 6번, 7번 인덱스 같이 배열을 완전 이진 트리 형태로 볼 수 있게 된다. 이러한 성질을 통해 특정 노드의 자식노드를 알 수도 있고, 반대로 한 노드의 부모노드를 알 수도 있게 된다.

힙 정렬이 이루어지기 위해서는 먼저 자료가 힙 상태를 띄어야 해서, 먼저 힙을 만드는 데서부터 시작을 해야 한다. 힙 생성에서는 주어진 배열을 최대 힙이나 최소 힙으로 만들어 주는데 (부모 노드의 키 값이 자식보다 큰 힙을 최대 힙, 작은 힙을 최소 힙이라고 한다) 쉬운 이해를 위해 트리 형식으로 설명을 하자면 주어진 아래의 배열을

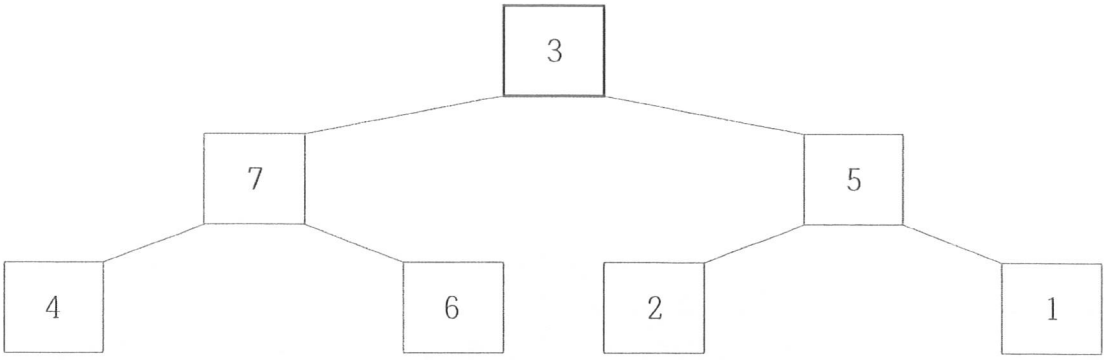
루트노드	3	7	1	4	6	5	2
0	1	2	3	4	5	6	7



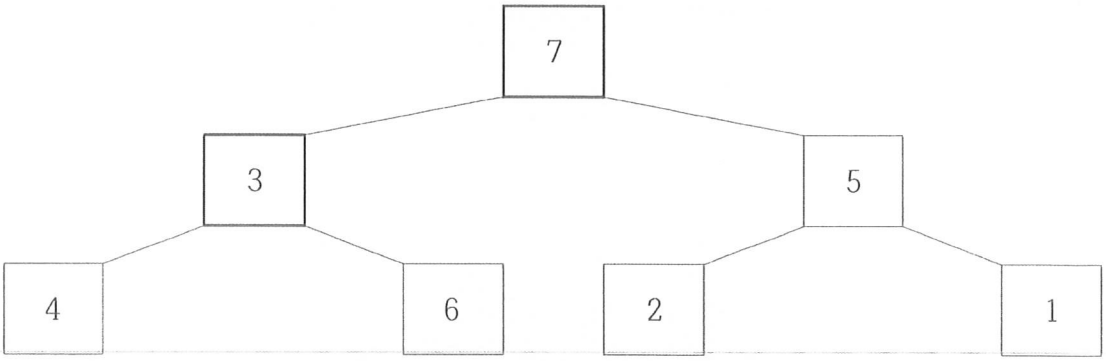
이와 같이 트리의 형태로 생각할 수 있다. 이를 최대힙 성질을 만족하도록 트리를 조정하려면 가장 말단의 자식을 가지는 노드와 그 서브 트리를 힙으로 만든다.



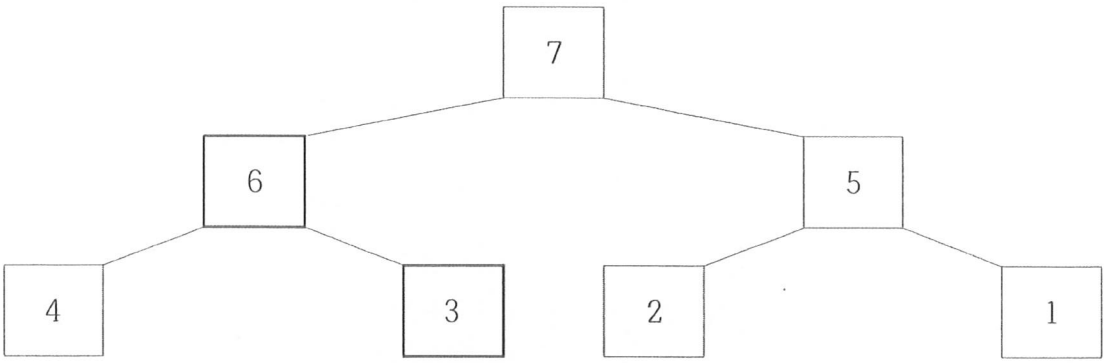
1의 자식 노드인 5와 2중에서 더 큰 값인 5와 1의 자리를 바꾼다. 그다음으로 자식을 가지는 노드 7은 이미 힙 상태를 띄고 있으므로 루트 노드인 3을 확인하면 된다.



3의 자식 노드인 7와 5중에서 더 큰 값인 7과 3의 자리를 바꾼다.



이 때 확인해야 할 사항으로 3이 7의 자리로 바뀌면서 부모 노드가 자식 노드보다 큰 값을 가지는 힙 상태를 만족하지 않고 있다. 따라서 바꾸게 된 이후에도 바꾼 노드를 힙 상태로 만들기 위해 재귀 호출을 한다.



마지막으로 3의 자식 노드인 4와 6중에서 더 큰 값인 6과 자리를 바꾸게 되면 해당 트리는 최대 힙 상태를 만족하게 되며 힙 정렬이 모두 끝나게 된다. 종료 당시 배열의 내용은 이러하다.

루트노드	7	6	5	4	3	2	1
0	1	2	3	4	5	6	7

추가로 도전 학기 동안 제작한 알고리즘들의 시간복잡도를 요약하여 표로 제작하였다.

알고리즘	최선	평균	최악
선택정렬	n^2	n^2	n^2
버블정렬	n^2	n^2	n^2
삽입정렬	n	n^2	n^2
병합정렬	nlogn	nlogn	nlogn
퀵정렬	nlogn	nlogn	n^2
힙정렬	nlogn	nlogn	nlogn

3. 도전 과제의 성과

도전 과제를 통해 19년도 2학기 때 배운 기본/고급 정렬 알고리즘에 대해 다시 한번 학습하며 내용을 토대로 충분히 다른 사람한테 설명할 수 있을 정도의 수준에 도달할 수 있게 되었다. 또한, 도전 과제를 진행하며 제작된 모든 정렬 알고리즘 코드들은 GitHub 사이트에 업로드되었고, 업로드된 내용은 모두가 열람이 가능한 열린 게시판으로 알고리즘이 궁금한 이들은 언제나 내용을 확인할 수 있다.

4. 자기평가

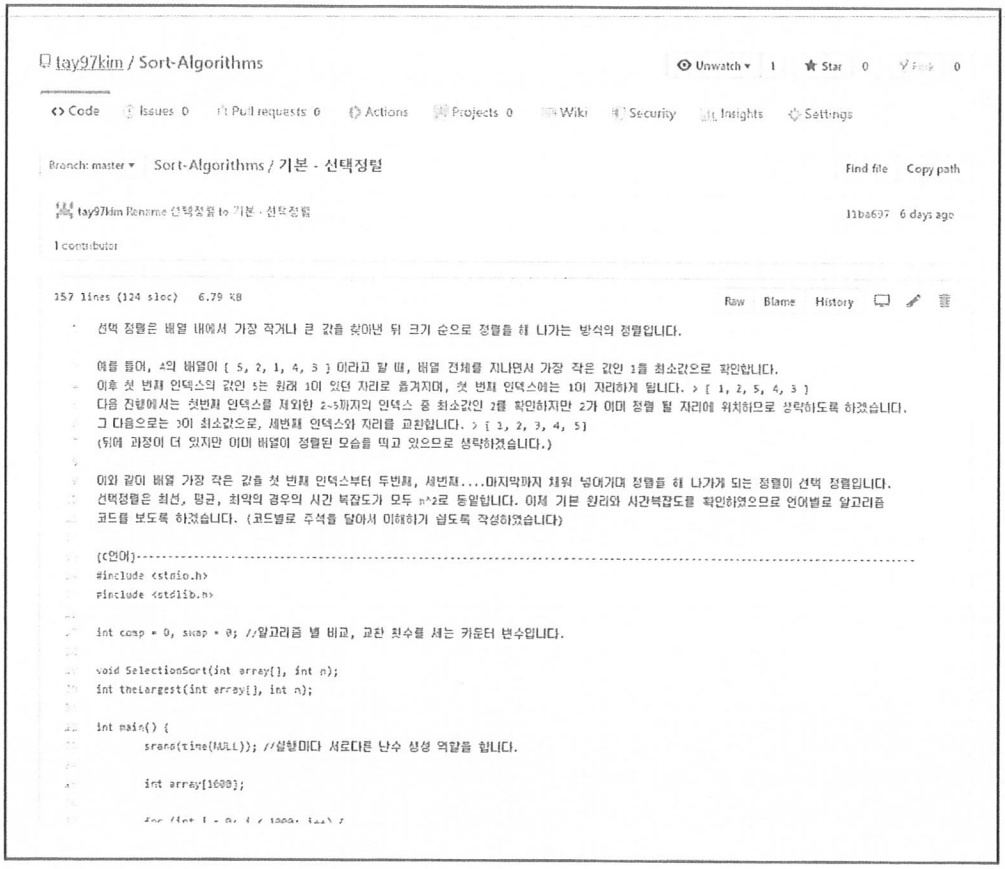
6주간의 짧은 기간 동안 세웠던 목표를 이루어 냈고, 또 필요한 부분에 대해서는 추가로 수정 및 보완하며 학습하고자 한 내용뿐만 아니라 여러 가지를 더 배우는 기회가 된 것 같다. 기존의 학교 수업이 아닌 자기만의 목표를 직접 정하는 과정에서 목표치를 너무 높게 잡은 것은 아닐까 걱정도 했었는데, 중간 보고서 제출 이후에도 하루하루 일정을 보내다 보니 또 다른 3주가 경과하였다. 다소 아쉬운 점은 여러 언어로 알고리즘 코드를 제작하게 되어 언어별 알고리즘 코드 세부사항을 자세히 짚지 못하며 진행한 것 같아서 도전 학기가 끝난 이후에도 관련 내용을 다시 보려 한다.

5. 최종 결과물

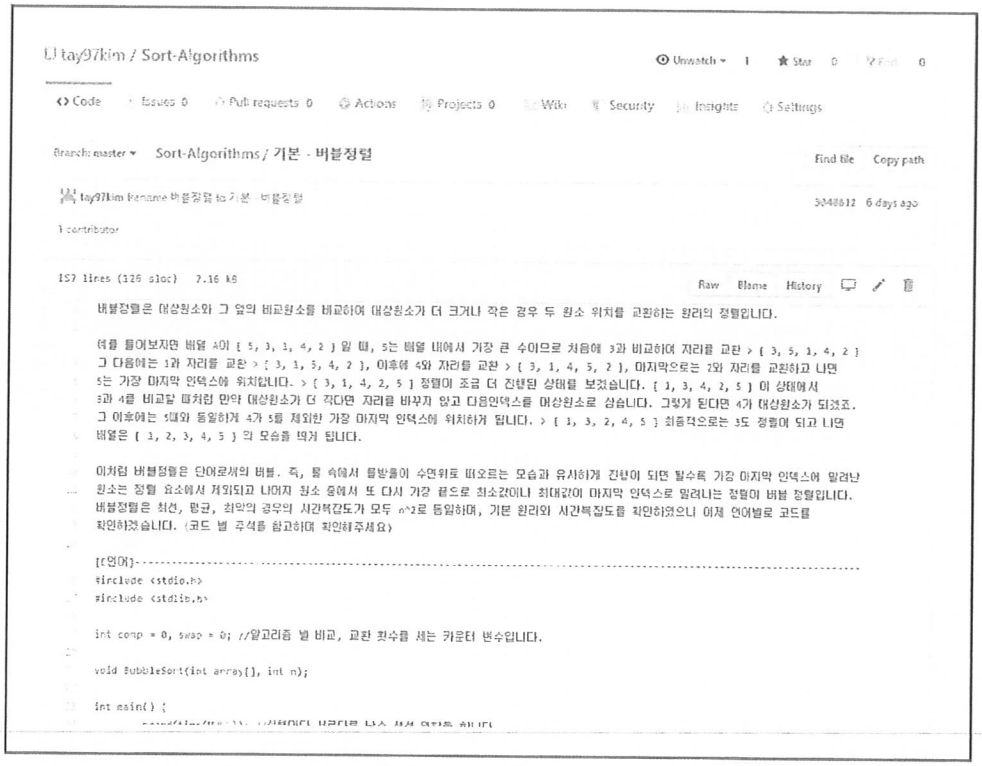
최종 결과물로 **제작된 모든 알고리즘 코드들을 제출** 및 **GitHub 사이트에 업로드**되었으며, 추가로 GitHub 사이트에서는 **정렬 알고리즘 별 특성 및 언어별 코드를 열람**할 수 있다.
(GitHub 사이트 주소: https://github.com/tay97kim/Sort-Algorithms)

[GitHub 사이트 메인화면] - 6가지 정렬 알고리즘에 대해 게시글이 있음

[각 게시글 세부 내용] - 정렬 알고리즘별 특성 및 작동원리, 언어별 코드가 있다.



(버블정렬 게시물)



tay97kim / Sort-Algorithms

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

Branch: master ▾ Sort-Algorithms / 기본 - 버블정렬 Find file Copy path

tay97kim Rename 버블정렬 to 기본 - 버블정렬 3048812 6 days ago

1 contributor

157 lines (126 sloc) 7.16 KB Raw Blame History

버블정렬은 대상원소와 그 옆의 비교원소를 비교하여 대상원소가 더 크거나 작은 경우 두 원소 위치를 교환하는 원리의 정렬입니다.

예를 들어보자면 배열 A이 [5, 3, 1, 4, 2] 일 때, 5는 배열 내에서 가장 큰 수이므로 처음에 3과 비교하여 자리를 교환 > [3, 5, 1, 4, 2]
그 다음에는 1과 자리를 교환 > [3, 1, 5, 4, 2], 이후에 4와 자리를 교환 > [3, 1, 4, 5, 2], 마지막으로는 2와 자리를 교환하고 나면
5는 가장 마지막 인덱스에 위치합니다. > [3, 1, 4, 2, 5] 정렬이 조금 더 진행된 상태를 보겠습니다. [1, 3, 4, 2, 5] 이 상태에서
3과 4를 비교할 때처럼 만약 대상원소가 더 작다면 자리를 바꾸지 않고 다음인덱스를 대상원소로 삼습니다. 그렇게 된다면 4가 대상원소가 되겠죠.
그 이후에는 5때와 동일하게 4가 5를 제외한 가장 마지막 인덱스에 위치하게 됩니다. > [1, 3, 2, 4, 5] 최종적으로는 3도 정렬에 되고 나면
배열은 [1, 2, 3, 4, 5] 의 모습을 띄게 됩니다.

이처럼 버블정렬은 단어로서의 버블. 즉, 물 속에서 물방울이 수면위로 떠오르는 모습과 유사하게 진행이 되면 될수록 가장 마지막 인덱스에 밀려난
원소는 정렬 요소에서 제외되고 나머지 원소 중에서 또 다시 가장 끝으로 최소값이나 최대값이 마지막 인덱스로 밀려나는 정렬이 버블 정렬입니다.
버블정렬은 최선, 평균, 최악의 경우의 시간복잡도가 모두 n^2로 동일하며, 기본 원리와 시간복잡도를 확인하였으니 이제 언어별로 코드를
확인하겠습니다. (코드 별 주석을 참고하며 확인해주세요)

[C언어]----------

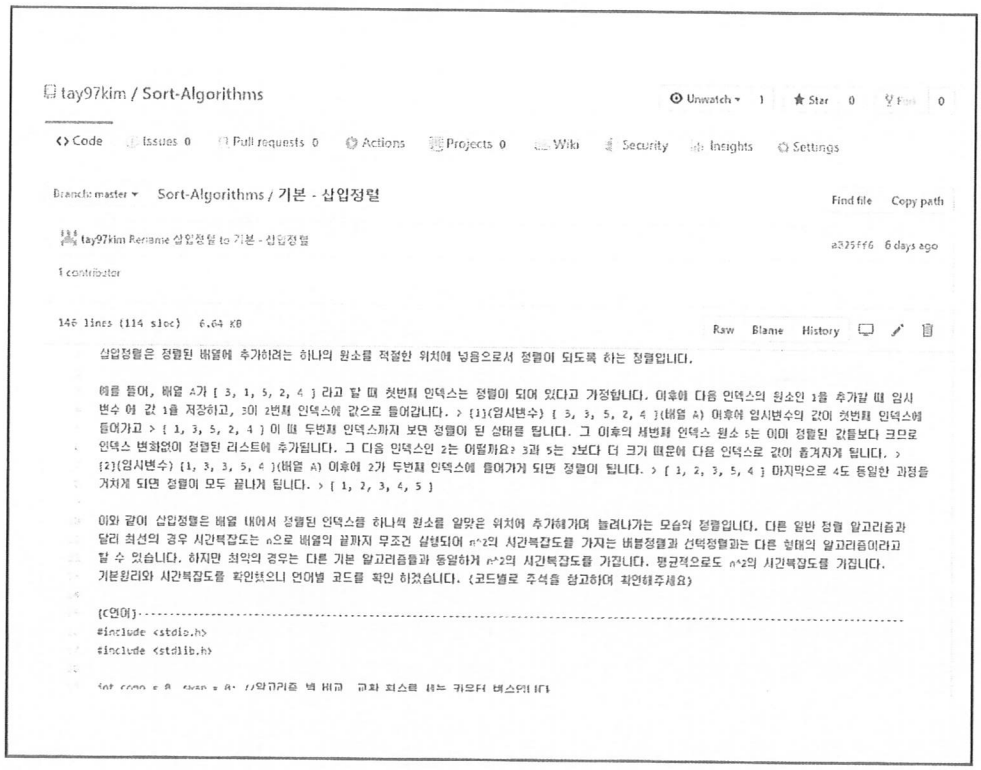
```
#include <stdio.h>
#include <stdlib.h>

int comp = 0, swap = 0; //알고리즘 별 비교, 교환 횟수를 세는 카운터 변수입니다.

void BubbleSort(int array[], int n);

int main() {
```

(삽입정렬 게시물)



tay97kim / Sort-Algorithms

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

Branch: master ▾ Sort-Algorithms / 기본 - 삽입정렬 Find file Copy path

tay97kim Rename 삽입정렬 to 기본 - 삽입정렬 a335ff6 6 days ago

1 contributor

146 lines (114 sloc) 6.64 KB Raw Blame History

삽입정렬은 정렬된 배열에 추가하려는 하나의 원소를 적절한 위치에 넣음으로서 정렬이 되도록 하는 정렬입니다.

예를 들어, 배열 A가 [3, 1, 5, 2, 4] 라고 할 때 첫번째 인덱스는 정렬이 되어 있다고 가정합니다. 이후에 다음 인덱스의 원소인 1을 추가할 때 임시
변수 에 값 1을 저장하고, 3이 2번째 인덱스에 값으로 들어갑니다. > (1)(임시변수) [3, 3, 5, 2, 4](배열 A) 이후에 임시변수의 값이 첫번째 인덱스에
들어가고 > [1, 3, 5, 2, 4] 이 때 두번째 인덱스까지 보면 정렬이 된 상태를 됩니다. 그 이후의 세번째 인덱스 원소 5는 이미 정렬된 값들보다 크므로
인덱스 변화없이 정렬된 리스트에 추가됩니다. 그 다음 인덱스인 2는 어떨까요? 3과 5는 2보다 더 크기 때문에 다음 인덱스로 값이 옮겨지게 됩니다. >
(2)(임시변수) [1, 3, 3, 5, 4](배열 A) 이후에 2가 두번째 인덱스에 들어가게 되면 정렬이 됩니다. > [1, 2, 3, 5, 4] 마지막으로 4도 동일한 과정을
거치게 되면 정렬이 모두 끝나게 됩니다. > [1, 2, 3, 4, 5]

이와 같이 삽입정렬은 배열 내에서 정렬된 인덱스를 하나씩 원소를 알맞은 위치에 추가해가며 늘려나가는 모습의 정렬입니다. 다른 일반 정렬 알고리즘과
달리 최선의 경우 시간복잡도는 n으로 배열의 끝까지 무조건 살펴되어 n^2의 시간복잡도를 가지는 버블정렬과 선택정렬과는 다른 형태의 알고리즘이라고
할 수 있습니다. 하지만 최악의 경우는 다른 기본 알고리즘들과 동일하게 n^2의 시간복잡도를 가집니다. 평균적으로도 n^2의 시간복잡도를 가집니다.
기본원리와 시간복잡도를 확인했으니 언어별 코드를 확인 하겠습니다. (코드별로 주석을 참고하며 확인해주세요)

[C언어]----------

```
#include <stdio.h>
#include <stdlib.h>

int comp = 0, swap = 0; //알고리즘 별 비교, 교환 횟수를 세는 카운터 변수입니다.
```

(퀵정렬 게시물)

tay97kim / Sort-Algorithms

Unwatch 1 Star 0

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

Branch: master Sort-Algorithms / 고급 - 퀵정렬 Find file Copy path

tay97kim Update 고급 - 퀵정렬 113c9bb 6 days ago

1 contributor

192 lines (153 sloc) 8.03 KB Raw Blame History

퀵정렬은 평균적인 성능이 좋아 가장 흔히 쓰이는 정렬 알고리즘입니다. 지정한 원소를 기준으로 그 원소보다 작은 그룹과 큰 그룹으로 나누어 재배치하며
각 그룹 또한 다시 그룹 내에서 한 원소를 지정하여 그 원소를 기준으로 값이 작은 그룹과 큰 그룹으로 재배치합니다.

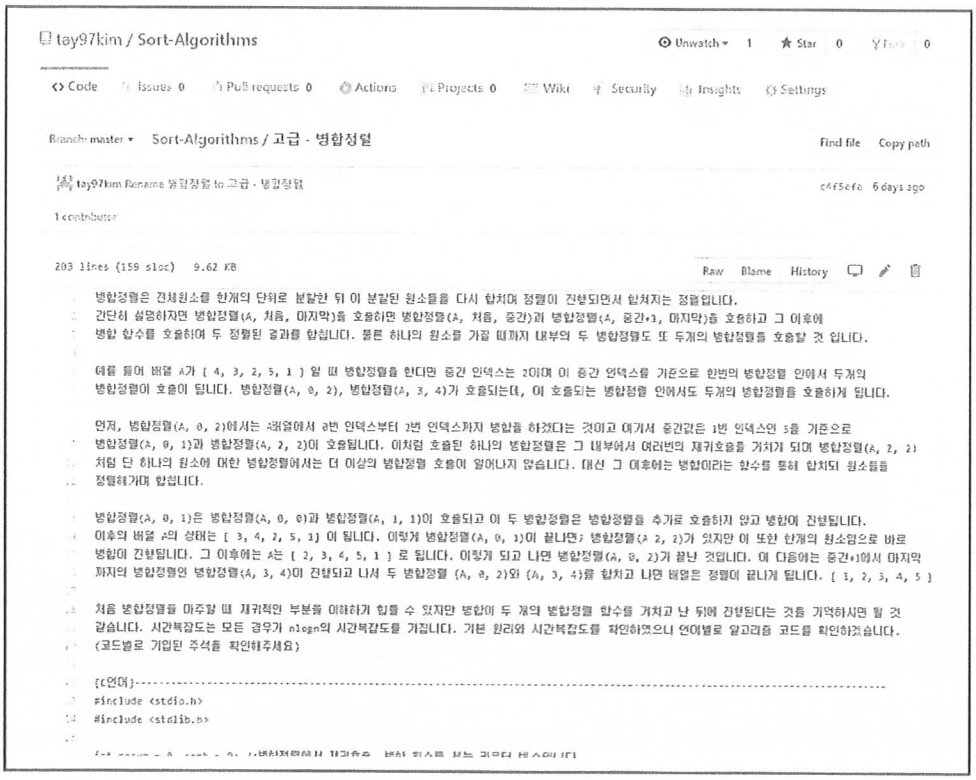
예를 들어, 배열 A가 [3, 1, 5, 2, 4]가 있을 때 기준원소를 제일 마지막 원소를 선택하면 값 4를 기준으로 작은 그룹 [3, 1, 2]과 큰 그룹 [5]이
생기게 됩니다. 작은 그룹 [3, 1, 2]에서도 원소 2를 기준으로 작은 그룹은 [1], 큰 그룹은 [3]으로 배치됩니다. 작은 그룹에 대해서는 정렬이 끝이
났으며, 큰 그룹인 [5]는 원소가 하나이므로 정렬이 끝나게 됩니다. [1, 2, 3, 4, 5]

다른 알고리즘에 비해 비교적 간단한 원리이기 때문에 고급 정렬 중에서는 이해하기 가장 쉬운 알고리즘에 속하다고 생각합니다. 시간복잡도는 평균적으로
nlogn의 시간복잡도를 가지지만, 이미 정렬되어 있는 최악의 상황에서는 n^2의 시간복잡도를 가지게 됩니다. 이제 언어별 알고리즘 코드를 확인해보겠습니다.
(코드별로 기입된 주석을 확인해주세요)

{c언어}----------

```
#include <stdio.h>
#include <stdlib.h>
```

(병합정렬 게시물)



tay97kim / Sort-Algorithms

Unwatch 1 Star 0

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

Branch: master Sort-Algorithms / 고급 - 병합정렬 Find file Copy path

tay97kim Rename 병합정렬 to 고급 - 병합정렬 c4f5efa 6 days ago

1 contributor

203 lines (159 sloc) 9.62 KB Raw Blame History

병합정렬은 전체원소를 한개의 단위로 분할한 뒤 이 분할된 원소들을 다시 합치며 정렬이 진행되면서 합쳐지는 정렬입니다.
간단히 설명하자면 병합정렬(A, 처음, 마지막)을 호출하면 병합정렬(A, 처음, 중간)과 병합정렬(A, 중간+1, 마지막)을 호출하고 그 이후에
병합 함수를 호출하여 두 정렬된 결과를 합칩니다. 물론 하나의 원소를 가질 때까지 내부의 두 병합정렬도 또 두개의 병합정렬을 호출할 것 입니다.

예를 들어 배열 A가 [4, 3, 2, 5, 1] 일 때 병합정렬을 한다면 중간 인덱스는 2이며 이 중간 인덱스를 기준으로 한번의 병합정렬 안에서 두개의
병합정렬이 호출이 됩니다. 병합정렬(A, 0, 2), 병합정렬(A, 3, 4)가 호출되는데, 이 호출되는 병합정렬 안에서도 두개의 병합정렬을 호출하게 됩니다.

먼저, 병합정렬(A, 0, 2)에서는 A배열에서 0번 인덱스부터 2번 인덱스까지 병합을 하겠다는 것이고 여기서 중간값은 1번 인덱스인 3을 기준으로
병합정렬(A, 0, 1)과 병합정렬(A, 2, 2)이 호출됩니다. 이처럼 호출된 하나의 병합정렬은 그 내부에서 여러번의 재귀호출을 거치게 되며 병합정렬(A, 2, 2)
처럼 단 하나의 원소에 대한 병합정렬에서는 더 이상의 병합정렬 호출이 일어나지 않습니다. 대신 그 이후에는 병합이라는 함수를 통해 합치되 원소들을
정렬해가며 합칩니다.

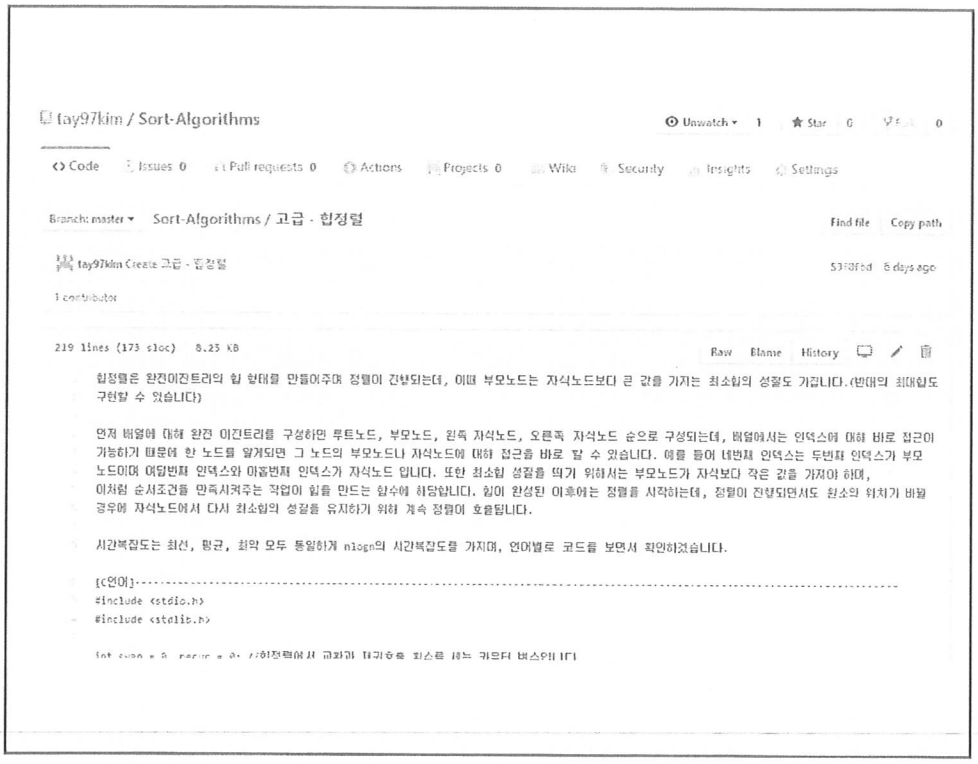
병합정렬(A, 0, 1)은 병합정렬(A, 0, 0)과 병합정렬(A, 1, 1)이 호출되고 이 두 병합정렬은 병합정렬을 추가로 호출하지 않고 병합이 진행됩니다.
이후의 배열 A의 상태는 [3, 4, 2, 5, 1] 이 됩니다. 이렇게 병합정렬(A, 0, 1)이 끝나면; 병합정렬(A 2, 2)가 있지만 이 또한 한개의 원소임으로 바로
병합이 진행됩니다. 그 이후에는 A는 [2, 3, 4, 5, 1] 로 됩니다. 이렇게 되고 나면 병합정렬(A, 0, 2)가 끝난 것입니다. 이 다음에는 중간+1에서 마지막
까지의 병합정렬인 병합정렬(A, 3, 4)이 진행되고 나서 두 병합정렬 (A, 0, 2)와 (A, 3, 4)를 합치고 나면 배열은 정렬이 끝나게 됩니다. [1, 2, 3, 4, 5]

처음 병합정렬을 마주할 때 재귀적인 부분을 이해하기 힘들 수 있지만 병합이 두 개의 병합정렬 함수를 거치고 난 뒤에 진행된다는 것을 기억하시면 될 것
같습니다. 시간복잡도는 모든 경우가 nlogn의 시간복잡도를 가집니다. 기본 원리와 시간복잡도를 확인하였으니 언어별로 알고리즘 코드를 확인하겠습니다.
(코드별로 기입된 주석을 확인해주세요)

{c언어}----------

```
#include <stdio.h>
#include <stdlib.h>
```

(힙정렬 게시물)



[작성 언어 종류] - 알고리즘별 C언어, Java, Python 총 세 가지로 작성되어 있다.
(C언어)

```
[c언어]----------
#include <stdio.h>
#include <stdlib.h>

int comp = 0, swap = 0; //알고리즘 별 비교, 교환 횟수를 세는 카운터 변수입니다.

void SelectionSort(int array[], int n);
int theLargest(int array[], int n);

int main() {
	srand(time(NULL)); //실행마다 서로다른 난수 생성 역할을 합니다.

	int array[1000];

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

		array[i] = rand() % 3000;
	} //1000 크기의 배열에 무작위 난수 원소 입력

	printf("[기존의 배열]\n");
	for (int i = 0; i < sizeof(array) / sizeof(int); i++) {
		printf("%d ", array[i]);
	} //기존의 배열을 모두 출력합니다.
	printf("\n");

	SelectionSort(array, sizeof(array) / sizeof(int));  // 선택정렬
```

(자바)

```
[Java]-------------------------------------------------------------------------------------------------------
public class BubbleSortTest { //클래스 이름을 동일하게 해주어야 합니다:

        static int comp = 0, swap = 0; //알고리즘 별 비교, 교환 횟수를 세는 카운터 변수입니다.

        static void swap(int[] array, int idx1, int idx2) {
                swap++;
                int t = array[idx1];
                array[idx1] = array[idx2];
                array[idx2] = t;
        } // swap 메소드는 배열의 두 인덱스 위치를 상호교환합니다.

        static void bubbleSort(int[] array, int n) {
                for (int i = 0; i < n - 1; i++) {
                        int changed = 0;
                        for (int j = n - 1; j > i; j--) {
                                comp++; //각 원소를 비교하는 for문에 의해 comp 카운터 증가
                                if (array[j - 1] > array[j]) {
                                        swap(array, j - 1, j); //swap 메소드 호출
                                        changed++;
                                }
                        }
                        if (changed == 0) break; // changed 변수가 0일 경우에는 값의 변경이 한번도 없는, 이미 정렬된 상태입니다.
                }
        }

        public static void main(String[] args) {
                int nx = 1000;  //배열 크기를 지정
                int[] array = new int[nx];

                for (int i = 0; i <= nx - 1; i++) {
                        array[i] = (int) (Math.random() * 3000) + 1;
                } //1000 크기의 배열에 무작위 난수 원소 입력

                System.out.print("기존의 배열:");
                for (int i = 0; i <= nx - 1; i++) {
                        System.out.print(" " + array[i]);
                } //기존의 배열을 모두 출력합니다.
                System.out.println("");
```

(파이썬)

```
[Python]----------------------------------------------------------------------------------------------------------
import random

def insertion_sort(my_list):
    global comp, swap #알고리즘 별 비교, 교환 횟수를 세는 카운터 변수입니다.
    comp = 0
    swap = 0
    my_list.insert(0, -1)
    for s_idx in range(2, len(my_list) ):
        temp = my_list[s_idx]
        ins_idx=s_idx
        while my_list[ins_idx-1] > temp:
            comp+=1 #각 원소를 비교하는 while문에 의해 comp 카운터 증가
            my_list[ins_idx] = my_list[ins_idx-1]
            ins_idx = ins_idx - 1

        my_list[ins_idx] = temp
        swap+=1 #임시변수로 옮겨진 새 원소가 정렬될 인덱스에 위치하며 swap 카운터 증가
    del my_list[0]


if __name__ == '__main__':
    list = []
    array = 1000
    for i in range(array):
        list.append(random.randint(1,array)) #1000 크기의 배열에 무작위 난수 원소 입력

    print("<기존의 배열>")
    print(list) #기존의 배열을 모두 출력합니다.

    insertion_sort(list) #삽입정렬
```


[참고문헌]

문병로 (2013) 쉽게 배우는 알고리즘 - 관계중심의 사고법: 한빛 아카데미

강민 (2018) 자료구조와 함께 배우는 알고리즘 입문 - C언어 편: 이지스 퍼블리싱

강민 (2018) 자료구조와 함께 배우는 알고리즘 입문 - 자바 편: 이지스 퍼블리싱

박선주 (2018) 필수 알고리즘 with 파이썬: 영진닷컴