
알 고 리 즘

Homework 1

Insertion sort, Merge sort

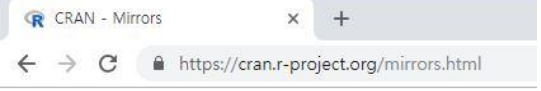

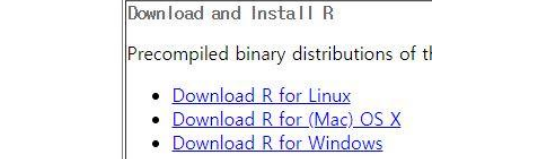


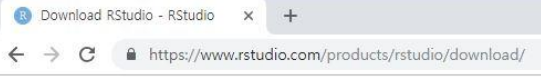
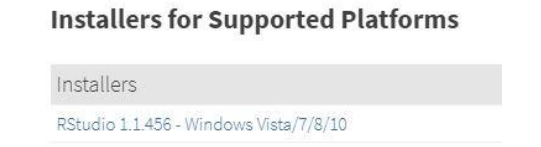



Professor	황호영 교수님
Department	컴퓨터공학과
Student ID	2012722028
Name	장 한 별
Date	2018. 10. 02.

Introduction.

이번 프로젝트의 목표는 Insertion Sort와 Merge Sort를 구현 후 이들의 성능을 비교해 보는 것이다. 먼저 θ -notation으로 time complexity를 분석하고, 다양한 언어를 이용해 programming 한다. 이후 best case, worst case, average case의 예를 입력해보고, 다양한 n에 대해 실행시간을 그래프로 나타낸다. 이후 두 Sort 알고리즘의 특징을 비교해보는 것을 목표로 한다.

To install R

	1. https://cran.r-project.org/mirrors.html
	2. Korea를 찾고, 맨 아래 url 클릭
	3. OS 환경에 따라 Download
	4. install R for the first time 클릭
	5. Download 클릭
	6. https://www.rstudio.com/products/rstudio/download/
	7. Installers 중 최신 버전 다운 하면 R 을 프로그래밍하고, 데이터 시각화를 사용할 수 있는 Rstudio 환경 구축 완료
	8. cat() 함수 안에 출력하고 싶은 문자열 삽입 가능 c언어에서 printf() 와 비슷 + 단일 문을 컴파일 후 실행 하는 것은 해당 문장을 클릭한 후 ctrl 키와 enter 키를 동시에 누르면 된다.

<pre>> cat("Hello world! \n") Hello world!</pre>	9. 위 문장을 실행 시켰을 때의 콘솔 화면이다.
<pre>3 x <- 1 4 y <- 2 5 x+y 6</pre>	10. 왼쪽 그림처럼 원하는 코드를 한번에 실행하고 싶을 경우, 원하는 코드를 모두 드래그 한후 ctrl 키와 enter 키를 동시에 누르면 된다.
<pre>> x <- 1 > y <- 2 > x+y [1] 3</pre>	11. 위 문장을 실행 시켰을 때의 콘솔 화면이다.
<pre>1 cat("Hello world! \n") 2 3 x <- 1 4 y <- 2 5 x+y 6</pre>	12. r 코드 내에 모든 코드를 한번에 컴파일 후 실행 시키고 싶을 경우, 10번과 같이 수행하면 된다. 이때 전체 코드를 드래그 하는 단축키인 ctrl 키와 a 키를 동시에 누르면 간편하다. 그 후 ctrl 키와 enter 키를 동시에 누르면 된다.
<pre>> cat("Hello world! \n") Hello world! > > x <- 1 > y <- 2 > x+y [1] 3</pre>	13. 위 문장을 실행 시켰을 때의 콘솔 화면이다.

1. Insertion Sort

For the insertion sort, consider that each input sequence has n different numbers. For the worst case, the average case, and the best case,

(a) Analyze the time complexity in θ -notation mathematically

2번째 배열의 요소부터 n 번째 배열의 요소까지 insertion을 위해 한 칸씩 옮겨 가며 sort 해야 한다. 이때의 중요한 것은 동일한 알고리즘으로 처음부터 끝까지 처리되어, 프로그래밍을 처리하는 시간이 모두 동일할 것이다. 따라서, 이 시간을 상수 $c(c > 0)$ 로 표현한다.

[Worst Case]

처음부터 끝까지 sort 알고리즘을 적용해야하는 경우. 즉, 역순으로 정렬된 경우.

$$\begin{aligned}
 T(n) &= c*1 + c*2 + c*3 + \dots + c*n \\
 &= c*(1+2+3+ \dots + n) \\
 &= c*(n-1)*(n-1+1)/2 \\
 &= cn^2/2 - cn/2
 \end{aligned}$$

Time complexity, $T(n)$ 을 θ -notation 으로 표기하면 $\theta(n^2)$ 이다.

[Average Case]

평균적으로 걸리는 시간이므로 대략적으로 Worst case의 절반이 걸린다고 가정한다.

$$\begin{aligned} T(n) &= (c*1 + c*2 + c*3 + \dots + c*n)/2 \\ &= (c*(1+2+3+ \dots + n))/2 \\ &= (c*(n-1)*(n-1+1)/2)/ \\ &= cn^2/4 - cn/4 \end{aligned}$$

Worst case 보다 2가 나누어진 것 같지만 계수를 무시하는 θ -notation 로 표기하면 average case 역시 $\theta(n^2)$ 이다.

[Best Case]

Best case는 이미 정렬이 모두 이루어진 상황이다. 이때는 배열의 길이 n 에 대해 좌우된다.

$$T(n) = c*n$$

Time complexity, $T(n)$ 을 θ -notation 으로 표기하면 $\theta(n)$ 이다.

(b) Write the program with your comments. Explain the program at least four lines.

```
6 # Insertion Sort 함수 정의 부분
7 InsertionSort <- function(array)
8 {
9   n <- length(array) # 정렬 하고자 하는 배열의 길이 저장
10  i <- 2; j <- 1      # 변수 초기화
11  key <- 0
12  while(1)
13  {
14    key = array[i] # key 값 변경
15    j = i - 1      # 위치 변경
16
17    while ((j > 0) && array[j] > key) # Insertion 위치 찾는 부분
18    {                                # key 보다 작은 값을 찾으면 Loop 중단
19      array[j + 1] = array[j];      # 배열 한 칸씩 옮겨가며 찾음
20      j = j - 1;
21    }
22
23    array[j+1] = key;                # 실제 Insertion
24    cat(i-1, ":", " ")              # 몇 번째 연산인지 출력
25    cat(array, "\n")
26
27    i = i + 1                        # 다음 배열 element 로 옮김
28
29    if(i == n + 1)                  # 배열의 길이 + 1 은 끝까지 다 갔다는 뜻
30      break
31  }
32  array                             # sort 된 배열 반환
33 }
```

위 그림은 Insertion Sort 함수를 정의한 부분이다. R에서 함수를 정의 할 때는 '함수의 이름 <- function(변수)' 형태로 선언해야 하는데, Insertion Sort 에서는 정렬하고 싶은 배열인 array 를 입

력해 Sort가 완료된 array를 반환해주는 형태로 함수를 선언했다. Length() 함수를 사용하여 배열의 길이를 n에 저장하고, 삽입 위치를 나타내는 각 변수들을 초기화 한다. 이때, 배열의 index를 주의 해야하는데 c언어를 포함한 많은 컴퓨터 언어는 0부터 시작하지만 R에서는 1부터 시작한다. 2번째 배열의 값을 key에 삽입해 비교하도록 한다. 기존 배열에서의 위치를 나타내는 j와 비교하여 key 보다작다면 한 칸씩 옮기며 찾도록 한다. 하나하나 옮기다보면 그 두 key 가 커지는 상황이 오는데 이때 key를 j의 바로 옆인 j+1 자리에 삽입한다. 모든 배열의 요소들을 이런식으로 하나하나 옮겨가며 sort 하도록한다. 배열의 길이 만큼에 다다르고, 그것보다 커지면 배열의 모든 요소가 sort가 완료되었다는 뜻이므로 빠져나온 후, 정렬된 배열을 반환 한다.

(c) For at least one example of each case, show the simulation results. Explain the simulation results at least four lines.

```
35 array1 <- c(1:8)           # best case
36 array2 <- c(8:1)           # worst case
37 array3 <- c(7,4,1,3,2,8,5,6) # average case
```

위 그림은 각 배열의 값들을 초기화한 부분이다. Insertion Sort 의 best case 일 때 배열의 요소들을 array1, worst case 일 때 배열의 요소들을 array2, average case 일 때 요소들을 array3 에 넣어줬다. (best case는 이미 정렬이 완료 되어 있는 case, worst case는 역순으로 정렬되어 있는 case, average case는 임의의 값을 무작위로 넣었을 때)

R에서 변수를 선언하고 초기화 할 때, 해당 변수를 '<-' 로 가리키고 넣어 주고 싶은 값을 써넣으면 된다. c() 함수는 보통 여러 개 값을 한번에 넣어 줄 때, 혹은 배열을 사용해야할 때 사용한다. 1:8 는 1, 2, 3, 4, 5, 6, 7, 8 을 모두 써줄 필요 없이 1~8 까지의 수를 순서대로 넣어 준다는 뜻이고 8:1 은 역순으로 넣어 준다는 뜻이다.

```
> n1 <- length(array1)
>
> tic("sleeping")           # 실행 시작 시간
> array1 <- InsertionSort(array1) # Insertion Sort 함수 호출
1 :      1 2 3 4 5 6 7 8
2 :      1 2 3 4 5 6 7 8
3 :      1 2 3 4 5 6 7 8
4 :      1 2 3 4 5 6 7 8
5 :      1 2 3 4 5 6 7 8
6 :      1 2 3 4 5 6 7 8
7 :      1 2 3 4 5 6 7 8
> cat("Output: ", array1, "\n")
Output: 1 2 3 4 5 6 7 8
> timeinfo<-toc()           # 실행 끝나는 시간
sleeping: 0.04 sec elapsed
> RT1 <- timeinfo$toc - timeinfo$tic # 끝나는 시간 - 시작 시간 = 실행시간 ;
> RT1
elapsed
0.04
```

위 그림은 best case 일 때 정렬 순서를 나타낸 화면이다. 이미 정렬이 다 되어 있는 상태이므로 가장 빠른 시간에 끝나는 best case 이다. 실행 시간은 약 0.04 초가 걸렸다.

```

> n2 <- length(array2)
>
> tic("sleeping")
> array2 <- InsertionSort(array2)
1 :      7 8 6 5 4 3 2 1
2 :      6 7 8 5 4 3 2 1
3 :      5 6 7 8 4 3 2 1
4 :      4 5 6 7 8 3 2 1
5 :      3 4 5 6 7 8 2 1
6 :      2 3 4 5 6 7 8 1
7 :      1 2 3 4 5 6 7 8
> cat("Output: ", array2, "\n")
Output: 1 2 3 4 5 6 7 8
> timeinfo<-toc()
sleeping: 0.12 sec elapsed
> RT2 <- timeinfo$toc - timeinfo$tic
> RT2
elapsed
  0.12

```

위 그림은 worst case 일 때 정렬 순서를 나타낸 화면이다. 역순으로 정렬되어 있는 배열을 하나하나 옮겨야 하므로 제일 많이 시간이 소요된다. 실행시간은 약 0.12 초가 걸렸다.

```

> n3 <- length(array3)
>
> tic("sleeping")
> array3 <- InsertionSort(array3)
1 :      4 7 1 3 2 8 5 6
2 :      1 4 7 3 2 8 5 6
3 :      1 3 4 7 2 8 5 6
4 :      1 2 3 4 7 8 5 6
5 :      1 2 3 4 7 8 5 6
6 :      1 2 3 4 5 7 8 6
7 :      1 2 3 4 5 6 7 8
> cat("Output: ", array3, "\n")
Output: 1 2 3 4 5 6 7 8
> timeinfo<-toc()
sleeping: 0.07 sec elapsed
> RT3 <- timeinfo$toc - timeinfo$tic
> RT3
elapsed
  0.07

```

위 그림은 average case 일 때 정렬 순서를 나타낸 화면이다. Key와 배열의 값을 확인 후 위치를 변경하는 부분과 Key를 확인만 하고 변경되지 않은 부분이 적절하게 섞여 있어 best case 의 시간과 worst case 사이의 시간으로 형성되었다. 실행시간은 약 0.07 초가 걸렸다.

(d) For the various values of n, show the graphs of running times for simulations. Write your simulation environment such as CPU and RAM.

Explain the graphs at least four lines.

```

1 install.packages("tictoc")
2 library(tictoc) # 실행시간 확인할 수 있는 라이브러리 추가

```

위 그림은 실행 시간을 확인할 수 있는 라이브러리인 "tictoc"을 추가 한 화면이다. 기본으로 설치되어 있는 라이브러리와 추가로 설치하려면 'install.packages("원하는 라이브러리")' 형식으로 미리 설치 후 사용해야한다. 한 번 설치한 후에는 주석으로 처리하여 다시 다운 받지 않아도 된다.

그 후 library() 함수를 이용해 "tictoc" 을 추가했다.

```
35 array1 <- c(8:1)      # 1~8 역순 정렬
36 array2 <- c(16:1)     # 1~16
37 array3 <- c(32:1)     # 1~32
38 array4 <- c(128:1)    # 1~128
39 array5 <- c(256:1)    # 1~256
40
```

위 그림은 각 배열들을 초기화한 화면이다. 필자는 n 이 8개, 16개, 32개, 128개, 256 개인 배열들로 총 5개의 배열 역순으로 저장하여 비교했다.

```
41 n1 <- length(array1)
42
43 tic("sleeping")      # 실행 시작 시간
44 array1 <- InsertionSort(array1) # Insertion Sort 함수 호출
45 cat("Output: ", array1, "\n")
46 timeinfo<-toc()      # 실행 끝나는 시간
47 RT1 <- timeinfo$toc - timeinfo$tic # 끝나는 시간 - 시작 시간 = 실행시간 ;
48 RT1
49 plot(n1, RT1, type="o",      # x 축 : n의 갯수, y 축 : 실행 시간
50      col="red", cex=2,      # 그래프 색 및 두께 설정
51      xlim = c(4,256), ylim = c(0,26), # x축, y축 범위 설정
52      xlab = "n", ylab="Running Time(s)")
53
```

위 그림은 첫번째 배열인 array1을 sort 시켜 실행시간을 출력하고 그래프에 그려 넣는 코드를 작성한 화면이다. Tic() 함수를 사용해 실행을 시작하는 시간을 저장하고 Toc() 함수를 사용해 실행이 끝나는 시간을 저장한다. Timeinfo 에서 toc 과 tic 을 빼주면 실행시간이 나오게 된다.

R 에서는 Plot 함수를 이용해 그래프를 그릴 수 있다. x 에는 배열의 길이, 즉 n의 개수, y 에는 running time 을 넣어줬다. 필자는 추가로 다양한 옵션을 추가했는데, 그래프의 색을 빨간색으로 설정하고, x축의 범위와 y축의 범위를 설정했다.

```
54 n2 <- length(array2)
55
56 tic("sleeping")
57 array2 <- InsertionSort(array2)
58 cat("Output: ", array2, "\n")
59 timeinfo<-toc()
60 RT2 <- timeinfo$toc - timeinfo$tic
61 RT2
62 points(n2, RT2, col="red", cex=2) # 점 추가
63 lines(c(n1,n2),c(RT1,RT2), col="red", cex=2) # 선 추가
64
```

위 그림은 두번째 배열인 array2를 sort 시켜 실행시간을 출력하고 그래프에 그려 넣는 코드를 작성한 화면이다. Array1에서 그래프를 처음 그릴 때와는 다르게 이번에는 points() 와 lines() 함수를 사용해 그래프를 그렸다.

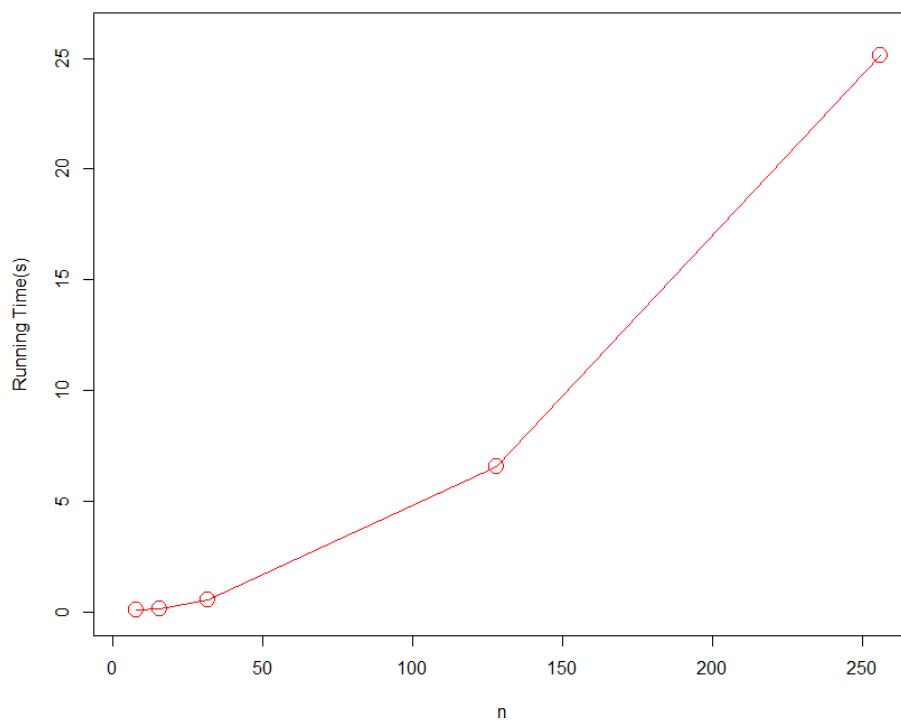
Plot() 함수는 그래프를 출력하는 부분을 모두 지우고 새롭게 그래프를 그리는 함수이므로 두 번째부터 plot() 사용하면 기존의 그래프가 모두 지워지고 새롭게 그려지게 된다. 이를 보완하기 위한 것이 lines() 함수이다. 이미 plot() 함수를 사용해 그래프를 그린 상태에서 lines() 함수를 그리면 기존에 있던 그래프에 추가로 그릴 수 있게 된다. Lines() 는 실선 그래프, points() 는 점 그래프를 의미한다. Array3, array4 도 이와 같이 작성했고, 마지막 배열인 array5까지 실행시킨다. 이때 x좌표와 y좌표값 부분에 n과 running time을 위치에 맞게 넣는 것이 중요하다.

```

86 n5 <- length(array5)
87
88 tic("sleeping")
89 array5 <- InsertionSort(array5)
90 cat("Output: ", array5, "\n")
91 timeinfo<-toc()
92 RT5 <- timeinfo$toc - timeinfo$tic
93 points(n5, RT5, col="red", cex=2)
94 lines(c(n4,n5),c(RT4,RT5), col="red", cex=2)
95

```

위 그림은 마지막 배열인 array5를 sort 시켜 실행시간을 출력하고 그래프에 그려 넣는 코드를 작성한 화면이다.



위의 모든 과정들을 수행하게 되면 위와 같은 그래프를 그릴 수 있게 된다. N의 개수가 작을 때는 별차이가 없어 보이지만, 128개가 되면서 눈에 띄게 증가하였고, 256개 일 때는 그래프의 기울기가 가파르게 올라가 sort 시 많은 시간이 소요됨을 확인할 수 있다. 필자는 1024개를 역순으로 정렬한 배열도 sort 하기 위해 시도 했지만, 너무 많은 시간이 소요되어 인내심의 한계에 봉착해 종료시켰다. 굳이 출력하지 않아도 많은 시간이 걸리는 것을 확인할 수 있었다.

시스템

프로세서: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.80 GHz
설치된 메모리(RAM): 8.00GB

위 그림은 필자의 PC 시스템 환경을 나타낸 화면이다. CPU 는 i7, Ram 은 8Gb 이다.

2. Merge Sort

For the merge sort, consider that each input sequence has n different numbers. For the worst case, the average case, and the best case,

(a) Analyze the time complexity in θ -notation mathematically

Root 에서는 배열의 길이가 n 인 배열이 1개 이므로 cn

그 아래 배열은 2개로 나누어져 배열의 길이가 $n/2$ 인 배열이 2개 이므로 $2*cn/2 = cn$

그 아래 배열은 다시 2개로 나누어져 배열의 길이가 $n/4$ 인 배열이 4개 이므로 $4*cn/4 = cn$

이런식으로 leave node 까지 구해서 배열의 길이가 1이 될 때 까지 나눈다면

배열의 길이가 1인 배열이 n 개이므로 $n*c = cn$ 이 된다.

그리고 Height h 는 $\log_2 n$, level 은 $\log_2 n + 1$ 이다.

[worst case]

각 층의 time complexity 는 cn 이 모든 층에서 발생하는 문제 일 때 높이만큼 곱해준다.

$$T(n) = cn * h$$

$$= cn * \log_2 n$$

이므로 θ -notation으로 표기하면 $\theta(n * \log n)$ 이다.

[best case]

이미 정렬이 모두 이루어진 배열일 때 Idea of master theorem 에서 case 2 인 경우 이다. 즉, leaves node 에 의해 θ -notation 이 결정되게 되고 모든 배열의 요소들이 1로 쪼개져 n 개의 개수를 가지므로 θ -notation으로 표기하면 $\theta(n)$ 이다.

[average case]

$$T(n) = 2 * T(n / 2) + cn$$

양변을 n 으로 나누면

$$T(n)/n = 2 * T(n / 2) / n + c$$

$$= T(n / 2) / (n / 2) + c$$

$$= T(n / 4) / (n / 4) + c + c$$

$$= T(n / 8) / (n / 8) + c + c + c$$

...

$$= T(n / n) / (n / n) + c + c + c + \dots + c$$

$$= \log_2 n$$

따라서,

$$T(n) = n * \log n$$

time complexity $T(n)$ 이 $n * \log n$ 이므로 θ -notation으로 표기하면 $\theta(n * \log n)$ 이다.

(b) Write the program with your comments. Explain the program at least four lines.

```
5 # Merge Sort 함수 정의 부분
6 MergeSort <- function(array){
7 {
8   n <- length(array) # 정렬 하고자 하는 배열의 크기 저장
9   if (n < 2)          # n 이 단 하나 밖에 없을 때 종료
10     return(array)    # 혹은 Divide 를 계속적으로 수행하여 1개 가 될때 까지
11
12   cat("[Divide] ")
13   first <- array[1:(n %% 2)] # 배열을 절반으로 나눠 앞 부분을 first 에 저장
14   cat(first, " | ")         # | 로 앞 뒤 구분
15   last <- array[(n %% 2 + 1):n] # 뒷 부분을 last 에 저장
16   cat(last, " ")
17   cat("\n")
18
19   first <- MergeSort(first) # 재귀적으로 수행하여
20   last <- MergeSort(last)  # divide를 더이상 수행 할수 없을 때 까지
21 }
```

Merge Sort는 크게 divide 부분과 conquer & combine 부분으로 크게 2가지로 나눌 수 있다. 위 그림은 그 중에서도 divide 부분을 나타낸 화면이다. 먼저 배열의 길이를 n 에 저장한다. n 이 만약 2보다 작다면 종료시키는데 이것은 배열의 값이 단 하나 일 때 혹은 Divide를 재귀적으로 수행하여 배열의 요소가 1개가 될 때까지 divide 했다는 의미이다.

R에서 ' $a \% b$ ' 연산자는 a 를 b 로 나누어 얻은 몫의 정수 값만 나타낼 수 있게 하는 연산자이다. N 을 2로 나누어 배열의 중간을 표시했고, 이 때 나누어진 절반의 앞부분을 first, 뒷부분을 last에 저장했다. 이를 다시 MergeSort 함수에 넣어 배열의 모든 요소가 1개로 나누어 질 때까지 재귀적으로 수행하도록 한다.

```
22 temp <- c() # 변수 초기화
23 i <- 0
24
25 while(length(first) > 0 && length(last) > 0) # 앞 뒤 배열의 값이 하나라도 남아 있으면 계속 실행
26 {
27   if(first[1] < last[1]) # 앞배열의 첫 번째 값이 뒷배열의 첫 번째 값보다 작을 때
28   {
29     temp[i + 1] <- first[1] # 앞배열의 첫 번째 값 삽입
30     first <- first[-1]     # 앞배열의 첫 번째 값 제거
31   }
32   else # 뒷배열의 첫 번째 값이 작으면
33   {
34     temp[i + 1] <- last[1] # 뒷배열의 첫 번째 값 삽입
35     last <- last[-1]      # 뒷배열의 첫 번째 값 제거
36   }
37   i = i + 1 # 배열의 다음 값으로 이동
38 }
39 temp <- c(temp, first, last) # 정렬할 배열에 다 넣기
40 cat("[Conquer] ")
41 cat(temp, "\n") # 정렬중인 배열 출력 후
42 temp # 반환
43 }
```

위 그림은 conquer & combine 부분의 화면이다. 변수를 초기화 후 while() 문 안으로 들어 간

다. 이때 배열의 앞부분과 뒷부분을 비교하는데 두 배열의 크기가 0보다 크면 계속 실행한다. 즉, 모든 배열의 요소가 정렬될 때까지 수행한다.

먼저 앞 배열의 첫 번째 값과 뒷 배열의 첫 번째 값을 비교해 값이 작은 배열의 값을 temp에 삽입 한다. 그 후 삽입된 배열의 값이 있는 배열의 첫 번째 값을 제거 후, 그 뒤의 요소를 첫 번째로 가리킨다. 결국 두 배열을 계속적으로 비교하여 작은 수가 먼저 temp 에 삽입되어 정렬된다. 모든 과정이 완료되었다면 temp를 반환한다.

(c) For at least one example of each case, show the simulation results. Explain the simulation results at least four lines.

```
46 array1 <- c(1:8)           # best case
47 array2 <- c(4,3,5,6,8,1,2,7) # worst case
48 array3 <- c(8:1)           # average case
49
```

위 그림은 각 배열의 값들을 초기화한 부분이다. Merge Sort 의 best case 일 때 배열의 요소들을 array1, worst case 일 때 배열의 요소들을 array2, average case 일 때 요소들을 array3 에 넣어줬다. (best case는 이미 정렬이 완료 되어 있는 case, worst case는 $O(n \cdot \log n)$ 인 case, average case는 역순으로 정렬된 case)

```
> n1 <- length(array1)
>
> tic("sleeping")           # 실행 시작 시간
> array1 <- MergeSort(array1)
[Divide] 1 2 3 4 | 5 6 7 8
[Divide] 1 2 | 3 4
[Divide] 1 | 2
[Conquer] 1 2
[Divide] 3 | 4
[Conquer] 3 4
[Conquer] 1 2 3 4
[Divide] 5 6 | 7 8
[Divide] 5 | 6
[Conquer] 5 6
[Divide] 7 | 8
[Conquer] 7 8
[Conquer] 5 6 7 8
[Conquer] 1 2 3 4 5 6 7 8
> cat("Output: ", array1, "\n")
Output: 1 2 3 4 5 6 7 8
> timeinfo<-toc()           # 실행 끝나는 시간
sleeping: 0.04 sec elapsed
> RT1 <- timeinfo$toc - timeinfo$tic # 끝나는 시간 - 시작 시간 = 실행시간 ;
> RT1
elapsed
0.04
> |
```

위 그림은 best case 일 때 정렬 순서를 나타낸 화면이다. 이미 정렬이 다 되어 있는 상태이므로 가장 빠른 시간에 끝나는 best case 이다. 각 배열의 요소가 반으로 divide 될 때, 앞 뒤 배열을 '|'로 구분했다. 실행시간은 약 0.04 초가 걸렸다.

```

> n2 <- length(array2)
>
> tic("sleeping")
> array2 <- MergeSort(array2)
[Divide] 4 3 5 6 | 8 1 2 7
[Divide] 4 3 | 5 6
[Divide] 4 | 3
[Conquer] 3 4
[Divide] 5 | 6
[Conquer] 5 6
[Conquer] 3 4 5 6
[Divide] 8 1 | 2 7
[Divide] 8 | 1
[Conquer] 1 8
[Divide] 2 | 7
[Conquer] 2 7
[Conquer] 1 2 7 8
[Conquer] 1 2 3 4 5 6 7 8
> cat("Output: ", array2, "\n")
Output: 1 2 3 4 5 6 7 8
> timeinfo<-toc()
sleeping: 0.05 sec elapsed
> RT2 <- timeinfo$toc - timeinfo$tic
> RT2
elapsed
0.05
>

```

위 그림은 worst case 일 때 정렬 순서를 나타낸 화면이다. Merge Sort 의 Worst Case는 $O(n \log n)$ 이 걸리는 배열의 값을 넣어준 후 정렬했다. 실행시간은 약 0.05 초가 걸렸다.

```

> n3 <- length(array3)
>
> tic("sleeping")
> array3 <- MergeSort(array3)
[Divide] 8 7 6 5 | 4 3 2 1
[Divide] 8 7 | 6 5
[Divide] 8 | 7
[Conquer] 7 8
[Divide] 6 | 5
[Conquer] 5 6
[Conquer] 5 6 7 8
[Divide] 4 3 | 2 1
[Divide] 4 | 3
[Conquer] 3 4
[Divide] 2 | 1
[Conquer] 1 2
[Conquer] 1 2 3 4
[Conquer] 1 2 3 4 5 6 7 8
> cat("Output: ", array3, "\n")
Output: 1 2 3 4 5 6 7 8
> timeinfo<-toc()
sleeping: 0.05 sec elapsed
> RT3 <- timeinfo$toc - timeinfo$tic
> RT3
elapsed
0.05
>

```

위 그림은 average case 일 때 정렬 순서를 나타낸 화면이다. 역순으로 정렬되어 있는 배열을 정렬했다. 실행시간은 약 0.05 초가 걸렸다. Merge sort 는 상당히 빠른 속도를 자랑하는 sort 방법 중에 속한다. N이 낮을 때는 Best case, Worst case, average case가 거의 동일한 것을 확인했고, 실제로는 PC 상황에 따라 오히려 Best case나 average case 일 때의 시간이 worst case 보다 오래 걸릴 수도 있는 것을 확인했다.

(d) For the various values of n, show the graphs of running times for simulations. Write your simulation environment such as CPU and RAM.

Explain the graphs at least four lines.

```
46 array1 <- c(8:1)      # 1~8 역순 정렬
47 array2 <- c(16:1)     # 1~16
48 array3 <- c(32:1)     # 1~32
49 array4 <- c(128:1)    # 1~128
50 array5 <- c(256:1)    # 1~256
51 |
```

위 그림은 각 배열들을 초기화한 화면이다. 필자는 n 이 8개, 16개, 32개, 128개, 256 개인 배열들로 총 5개의 배열 역순으로 저장하여 비교했다.

```
52 n1 <- length(array1)
53
54 tic("sleeping")      # 실행 시작 시간
55 array1 <- MergeSort(array1)
56 cat("Output: ", array1, "\n")
57 timeinfo<-toc()      # 실행 끝나는 시간
58 RT1 <- timeinfo$toc - timeinfo$tic # 끝나는 시간 - 시작 시간 = 실행시간 ;
59 RT1
60 plot(n1, RT1, type="o",      # x 축 : n의 갯수, y 축 : 실행 시간
61      col="blue", cex=2,      # 그래프 색 및 두께 설정
62      xlim = c(4,256), ylim = c(0,26), # x 축, y 축 범위 설정
63      xlab = "n", ylab="Running Time(s)")
64
```

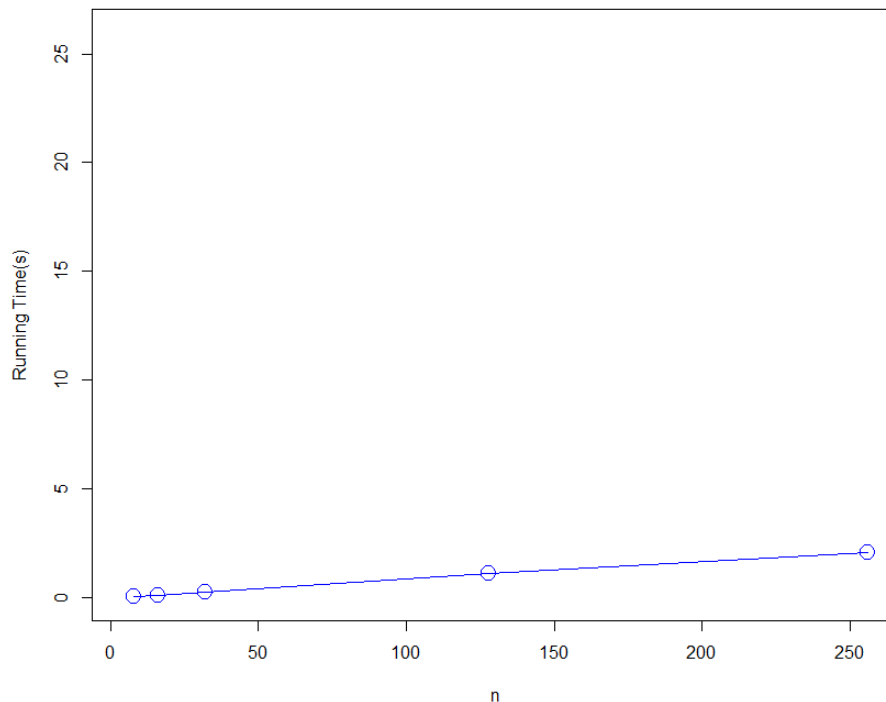
위 그림은 첫번째 배열인 array1을 sort 시켜 실행시간을 출력하고 그래프에 그려 넣는 코드를 작성한 화면이다. Insertion Sort 코드와 거의 동일하다. Plot() 함수를 이용해 그래프를 그리는 부분에서 그래프 색을 변경하는 옵션인 col 에 "blue"를 넣어 그래프의 색을 파란색으로 설정했다. 이는 Insertion Sort 와의 비교를 위해 설정한 것이다.

```
65 n2 <- length(array2)
66
67 tic("sleeping")
68 array2 <- MergeSort(array2)
69 cat("Output: ", array2, "\n")
70 timeinfo<-toc()
71 RT2 <- timeinfo$toc - timeinfo$tic
72 RT2
73 points(n2, RT2, col="blue", cex=2) # 점 추가
74 lines(c(n1,n2),c(RT1,RT2), col="blue", cex=2) # 선 추가
75
```

위 그림은 두번째 배열인 array2를 sort 시켜 실행시간을 출력하고 그래프에 그려 넣는 코드를 작성한 화면이다. Array1에서 그래프를 처음 그릴 때 plot() 을 사용한 것과는 다르게 이번에는 points() 와 lines() 함수를 사용해 그래프를 그렸다.

```
97 n5 <- length(array5)
98
99 tic("sleeping")
100 array5 <- MergeSort(array5)
101 cat("Output: ", array5, "\n")
102 timeinfo<-toc()
103 RT5 <- timeinfo$toc - timeinfo$tic
104 points(n5, RT5, col="blue", cex=2)
105 lines(c(n4,n5),c(RT4,RT5), col="blue", cex=2)
```

위 그림은 마지막 배열인 array5를 sort 시켜 실행시간을 출력하고 그래프에 그려 넣는 코드를 작성한 화면이다.



위의 모든 과정들을 수행하게 되면 위와 같은 그래프를 그릴 수 있게 된다. Insertion Sort 와 비교해 그래프의 기울기가 훨씬 완만한 형태를 보인다. 이는 Merge Sort가 많은 Sort 종류 중에서 상당히 빠른 속도를 자랑하는 sort 방법임을 알 수 있다. 256 개가 모두 정렬되는 시간은 3초 가 채 되지않았다.

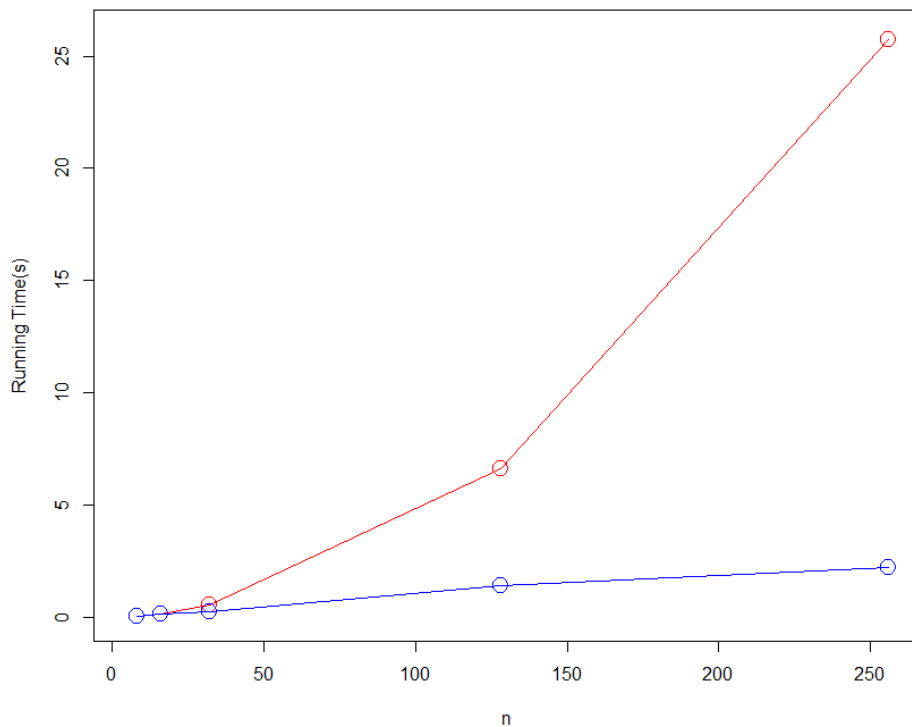
(e) Compare the graphs with those of the insertion sort at least four lines.

```

52 n1 <- length(array1)
53
54 tic("sleeping")           # 실행 시작 시간
55 array1 <- MergeSort(array1)
56 cat("Output: ", array1, "\n")
57 timeinfo<-toc()          # 실행 끝나는 시간
58 RT1 <- timeinfo$toc - timeinfo$tic # 끝나는 시간 - 시작 시간 = 실행시간 ;
59 RT1
60 points(n1, RT1, type="o",           # x 축 : n의 갯수, y 축 : 실행 시간
61        col="blue", cex=2,           # 그래프 색 및 두께 설정
62        xlim = c(4,256), ylim = c(0,26), # x축, y축 범위 설정
63        xlab="n", ylab="Running Time(s)")
64

```

위 그림은 Insertion Sort와 Merge Sort 의 실행 시간을 비교하기 위해 수정한 코드의 화면이다. Plot() 으로 그래프를 그린후 points() 나 lines() 으로 추가하면 된다는 사실을 이미 기술했으므로, Insertion Sort 의 모든 코드를 실행시킨 후, Merge Sort 의 첫 번째 배열 부분에서 Plot() 을 points() 로 수정해 Insertion Sort 그래프 와 Merge Sort 그래프를 겹쳐 그릴 수 있도록 한다.



위 그림은 Insertion Sort 의 실행시간 그래프와 Merge Sort 실행 시간 그래프를 겹쳐 그린 화면이다. 빨간색 그래프는 insertion sort의 그래프이고, 파란색 그래프는 merge sort의 그래프를 나타낸다. 이렇게 비교함으로써 n 이 증가할수록 두 실행시간의 차이가 기하급수적으로 차이가 남을 시각적으로 확인할 수 있다. 이론적으로 n 이 작은 수 일때는 insertion sort가 약간 더 빠르거나 거의 비슷하다. 이를 그래프를 통해 다시 한 번 확인할 수 있었고, 특히 n 이 256 일 때, 실행시간은 약 10배 차이가 난다는 것을 확인하여, n 이 증가함에 따라 이 증가 폭은 훨씬 증가할 것으로 예상된다.

Conclusion.

이번 프로젝트는 Insertion Sort 와 Merge Sort 를 직접 구현해보고, 배열의 길이가 증가함에 따라 실행시간이 어떻게 변하는지 그래프로 나타내는 프로젝트였다. Insertion Sort 는 하나의 key 값과 배열의 요소들을 비교해 하나하나 위치를 변경하면서 정렬하는 Sort 알고리즘이다. N 이 작으면 적당한 속도로 수행이 이루어 지는 반면, n 이 크면 정말 느리다는 단점이 있다. 하지만 그만큼 구현이 간단하다는 장점이 있다. 실제로 필자가 이를 구현하는 것은 일도 아니었다.

Merge Sort는 배열의 모든 요소들을 하나로 divide한 후 낮은 값부터 차례로 배열에 담은 후, 정렬

하는 Sort 알고리즘이다. 필자는 Merge Sort를 구현해 본적이 없어서 처음 시작부터 문제가 많았다. 이미 나누어져 정렬된 두 배열에서 작은 수부터 차례로 정렬하는 부분은 그림을 통해 쉽게 이해할 수 있었지만, 처음부터 어떻게 그 과정까지 이루어졌는지에 대한 정보는 쉽게 이해가 가지 않았다. 서적을 참고하고 많은 자료들을 읽고 복습한 후 어느 정도 감이 잡혀 직접 구현해 이를 해결할 수 있었다.

가장 큰 문제는 이미 하나로 묶여있는 배열의 모든 요소들을 divide 해 하나로 만들어 내는 재귀적으로 함수를 호출하는 부분이었다. 이 부분은 보다 디버깅 작업이 쉬운 C언어로 먼저 코딩해서 해결했다. 배열을 계속 나누는 부분을 단순히 2로 나누어 주는 것보다 재귀 함수를 호출하여 해결하는 것이 훨씬 효율적이고, 이해도도 훨씬 높아졌다. 이를 토대로 R에 적용시켜 구현할 수 있었다.

Merge Sort 는 이처럼 구현이 어려운 알고리즘에 속하지만 속도면에서 성능 하나만큼은 정말 좋다. 실제로 필자가 n 이 256 개로 실험했을 때 3초도 걸리지 않는 결과를 확인했고, 이는 Insertion Sort 에 비해 약 10배가 빠르다. N 이 증가할수록 이 성능의 차이는 더욱 기하급수적으로 벌어지는 것 또한 예상할 수 있다. 하지만, merge sort는 배열을 divide 해서 따로 저장해야 하므로 보통 추가적인 space를 많이 차지하는 단점이 있다. 모든 경우에서 항상 최고인 경우는 없으니 Time complexity 와 space complexity를 상황에 맞게 대입하여 판단하는 것이 매우 중요하다고 생각한다.

이번 과제를 통해 많은 sort 알고리즘을 접할 수 있었고, Bubble Sort, Quick Sort 등 다양한 sort 를 모두 직접 구현해 보고 비교해 보고싶다는 목표가 생겼다.

Reference.

<https://www.statmethods.net/graphs/line.html>

<https://code.i-harness.com/ko-kr/q/770c05>

<http://wkdgusdn3.tistory.com/entry/MergeSort%EB%B3%91%ED%95%A9%EC%A0%95%EB%A0%AC>

https://en.wikipedia.org/wiki/Insertion_sort#cite_note-5

https://en.wikipedia.org/wiki/Merge_sort

<http://rstatistics.tistory.com/3#points>

<http://the-r.kr/2017/05/30/5-ways-to-measure-running-time-of-r-code-r-bloggers/>

열혈강의 자료구조/ 이상진/ 프리렉

Do it! 쉽게 배우는 R 데이터 분석/ 김영우/ 이지스퍼블리싱