알고리즘 HW1

공과대학 컴퓨터공학부
2021-16988 박재완

1번

(1) Master Thm을 이용하면, $a = 4, b = 4, f(n) = b, h(n) = n$이다. $\dfrac{f(n)}{h(n)} = \dfrac{b}{n} = O\left(\dfrac{1}{n}\right)$이므로, $T(n) = \Theta(n)$.

(2) $n = 3^k$라고 가정하자. 그러면 다음과 같다.

$$\begin{aligned}
T(n) &= 3T\left(\frac{n}{3}\right) + n\log n = 3T\big(3^{k-1}\big) + 3^k \log 3^k \\
&= 3\big(3T\big(3^{k-2}\big) + 3^{k-1}\log 3^{k-1}\big) + 3^k\log 3^k = 3^2T\big(3^{k-2}\big) + 3^k\big(\log 3^k + \log 3^{k-1}\big) \\
&= \cdots = 3^kT\big(3^0\big) + 3^k\big(\log 3^k + \cdots + \log 3^1\big) \\
&= 3^kT(1) + \frac{\log 3}{2}3^k k(k+1) = nT(1) + \frac{\log 3}{2} n\log_3 n(\log_3 n + 1)
\end{aligned}$$

따라서 $T(n) = \Theta\Big(n(\log n)^2\Big)$

(3) Master Thm을 이용하면, $a = 5,\ b = 5,\ f(n) = 3n,\ h(n) = n$이다. $\dfrac{f(n)}{h(n)} = 3 = \Theta(1)$이므로, $T(n) = \Theta(n\log n)$.

(4) 어떤 양의 상수 c가 존재하여 $k < n$인 모든 k에 대하여 $T(k) \le ck\log k$을 만족한다고 가정하자. 그러면 충분히 큰 n에 대하여 다음이 성립한다.

$$\begin{aligned}
T(n) &= T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + \Theta(n) \\
&\le \frac{cn}{4}\log\left(\frac{n}{4}\right) + \frac{3cn}{4}\log\left(\frac{3n}{4}\right) + \Theta(n) = \frac{cn}{4}\log\left(\frac{n}{4}\right) + \frac{3cn}{4}\left(\log 3 + \log\left(\frac{n}{4}\right)\right) + \Theta(n) \\
&= cn\log\left(\frac{n}{4}\right) + \frac{3\log 3}{4}cn + \Theta(n) = cn\log n + \log\frac{3^{\frac{3}{4}}}{4}cn + \Theta(n) \\
&\le cn\log n + \log\frac{3^{\frac{3}{4}}}{4}cn + dn \quad (\exists d > 0) \\
&\le cn\log n \quad \left(\left(\frac{4}{3^{\frac{3}{4}}}\right)^c > d \text{ 인 } c \text{ 에 대하여 성립}\right)
\end{aligned}$$

따라서 충분히 큰 모든 정수 n에 대하여 $T(n) \le cn\log n$이 성립한다고 할 수 있다. 따라서 $T(n) = O(n\log n)$.

어떤 양의 상수 c가 존재하여 $k < n$인 모든 k에 대하여 $T(k) \ge ck\log k$을 만족한다고 가정하자. 그러면

충분히 큰 n에 대하여 다음이 성립한다.

$$\begin{aligned}
T(n) &= T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + \Theta(n) \\
&\geq \frac{cn}{4}\log\left(\frac{n}{4}\right) + \frac{3cn}{4}\log\left(\frac{3n}{4}\right) + \Theta(n) = cn\log n + \log\frac{3^{\frac{3}{4}}}{4}cn + \Theta(n) \\
&\geq cn\log n + \log\frac{3^{\frac{3}{4}}}{4}cn + dn \ \ (\exists d > 0) \\
&\geq cn\log n \ \ \left(\left(\frac{4}{3^{\frac{3}{4}}}\right)^c < d \text{ 인 } c \text{ 에 대하여 성립}\right)
\end{aligned}$$

따라서 충분히 큰 모든 정수 n에 대하여 $T(n) \geq cn\log n$이 성립한다고 할 수 있다. 따라서 $T(n) = \Omega(n\log n)$.

종합적으로 $T(n) = \Theta(n\log n)$이다.

(5) 어떤 양의 상수 c가 존재하여 $k < n$인 모든 k에 대하여 $T(k) \leq ck\log k$을 만족한다고 가정하자. 그러면 충분히 큰 n과 $c = 5$에 대하여 다음이 성립한다.

$$\begin{aligned}
T(n) &= 3T\left(\frac{n}{3} + 9\right) + n \\
&\leq 3c\left(\frac{n}{3} + 9\right)\log\left(\frac{n}{3} + 9\right) + n = c(n + 27)\log\left(\frac{n}{3} + 9\right) + n \\
&\leq c(n + 27)\log\frac{4n}{5} + n \\
&= cn\log n + \left(c\log\frac{4}{5} + 1\right)n + 27c\log\frac{4n}{5} \\
&\leq cn\log n
\end{aligned}$$

따라서 충분히 큰 모든 정수 n에 대하여 $T(n) \leq cn\log n$이 성립한다고 할 수 있다. 따라서 $T(n) = O(n\log n)$.

어떤 양의 상수 c가 존재하여 $k < n$인 모든 k에 대하여 $T(k) \geq ck\log k$을 만족한다고 가정하자. 그러면 충분히 큰 n과 $c = 1$에 대하여 다음이 성립한다.

$$\begin{aligned}
T(n) &= 3T\left(\frac{n}{3} + 9\right) + n \\
&\geq 3c\left(\frac{n}{3} + 9\right)\log\left(\frac{n}{3} + 9\right) + n = c(n + 27)\log\left(\frac{n}{3} + 9\right) + n \\
&\geq c(n + 27)\log\frac{n}{3} + n \\
&= cn\log n + \left(c\log\frac{1}{3} + 1\right)n + 27c\log\frac{n}{3} \\
&\geq cn\log n
\end{aligned}$$

따라서 충분히 큰 모든 정수 n에 대하여 $T(n) \geq cn\log n$이 성립한다고 할 수 있다. 따라서 $T(n) = \Omega(n\log n)$.

종합적으로 $T(n) = \Theta(n\log n)$이다.

2번

함수 `sample(A[], p, r)`의 실행시간을 $T(p,r)$이라 하자. 그러면 $r-p>3$인 경우, `if` 블럭에서 1번의 비교, `for` 블럭에서 $(r-p+1)$번의 비교를 하게 된다. 이후 `sample(A[], p, p+q-1)`, `sample(A[], p+2q, r)`을 실행한다. 따라서 실행시간은 다음과 같다.

$$\begin{aligned} T(p,r) &= 1+(r-p+1)+T\left(p, p+\left\lfloor\frac{r-p+1}{4}\right\rfloor-1\right)+T\left(p+2\left\lfloor\frac{r-p+1}{4}\right\rfloor, r\right) \\ &= r-p+2+T\left(p, p+\left\lfloor\frac{r-p+1}{4}\right\rfloor-1\right)+T\left(p+2\left\lfloor\frac{r-p+1}{4}\right\rfloor, r\right) \end{aligned}$$

어떤 양의 상수 c가 존재하여 $r'-p'+1<r-p+1$인 모든 p', r'에 대하여 $T(p',r')\le c(r'-p'+1)$을 만족한다고 가정하자. 그러면 $(r-p+1)$이 충분히 클 때, 다음이 성립한다.

$$\begin{aligned} T(p,r) &= r-p+2+T\left(p, p+\left\lfloor\frac{r-p+1}{4}\right\rfloor-1\right)+T\left(p+2\left\lfloor\frac{r-p+1}{4}\right\rfloor, r\right) \\ &\le r-p+2+c\left\lfloor\frac{r-p+1}{4}\right\rfloor+c\left(r-p+1-2\left\lfloor\frac{r-p+1}{4}\right\rfloor\right) \\ &= (c+1)(r-p+1)-c\left\lfloor\frac{r-p+1}{4}\right\rfloor+1 \\ &= c(r-p+1)+\left((r-p+1)-c\left\lfloor\frac{r-p+1}{4}\right\rfloor+1\right) \\ &\le c(r-p+1) \end{aligned}$$

$r-p+1\ge 4$일 때, $c=8$을 선택해주면 $(r-p+1)-c\left\lfloor\frac{r-p+1}{4}\right\rfloor+1\le 0$이 되어 위 관계가 성립한다. 따라서 충분히 큰 $(r-p+1)$에 대하여 $T(p,r)\le c(r-p+1)$이 성립한다고 할 수 있다. 따라서 $T(p,r)=O(r-p)$.

어떤 양의 상수 c가 존재하여 $r'-p'+1<r-p+1$인 모든 p', r'에 대하여 $T(p',r')\ge c(r'-p'+1)$을 만족한다고 가정하자. 그러면 $(r-p+1)$이 충분히 클 때, 다음이 성립한다.

$$\begin{aligned} T(p,r) &= r-p+2+T\left(p, p+\left\lfloor\frac{r-p+1}{4}\right\rfloor-1\right)+T\left(p+2\left\lfloor\frac{r-p+1}{4}\right\rfloor, r\right) \\ &\ge r-p+2+c\left\lfloor\frac{r-p+1}{4}\right\rfloor+c\left(r-p+1-2\left\lfloor\frac{r-p+1}{4}\right\rfloor\right) \\ &= c(r-p+1)+\left((r-p+1)-c\left\lfloor\frac{r-p+1}{4}\right\rfloor+1\right) \\ &\ge c(r-p+1) \end{aligned}$$

$c=1$을 선택해주면 $(r-p+1)-c\left\lfloor\frac{r-p+1}{4}\right\rfloor+1\ge 0$이 되어 위 관계가 성립한다. 따라서 $(r-p+1)$이 충분히 큰 모든 p, r에 대하여 $T(p,r)\ge c(r-p+1)$이 성립한다고 할 수 있다. 따라서 $T(p,r)=\Omega(r-p)$.

종합적으로 $T(p,r)=\Theta(r-p)$이다.

3번

함수 `test(n)`의 실행시간을 $T(n)$이라 하자. 그러면 `if` 블럭에서 1번의 비교를 하게 되고, $n > 50$인 경우 `test(n/3 + 5)`와 `test(2n/3 + 7)`을 실행한다. 따라서 실행시간은 다음과 같다.

$$T(n) = 1 + T\left(\frac{n}{3} + 5\right) + T\left(\frac{2n}{3} + 7\right)$$

어떤 양의 상수 c가 존재하여 $k < n$인 모든 k에 대하여 $T(k) \leq ck - 2$을 만족한다고 가정하자. 그러면 n이 충분히 클 때, 다음이 성립한다.

$$\begin{aligned} T(n) &= 1 + T\left(\frac{n}{3} + 5\right) + T\left(\frac{2n}{3} + 7\right) \\ &\leq 1 + c\left(\frac{n}{3} + 5\right) - 2 + c\left(\frac{2n}{3} + 7\right) - 2 \\ &= cn + 12c - 3 \\ &\leq cn - 2 \end{aligned}$$

$c = \dfrac{1}{12}$을 선택해주면 $12c - 3 = -2$가 되어 위 관계가 성립한다. 따라서 충분히 큰 모든 n에 대하여 $T(n) \leq cn - 2$이 성립한다고 할 수 있다. 따라서 $T(n) = O(n)$.

4번

길이가 1인 배열에 대하여 selection sort 알고리즘이 성공적으로 sorting을 함은 자명하다. 길이가 n인 배열에 대하여 알고리즘이 성공적인 sorting을 한다고 가정하자. 재귀적 selection sort 알고리즘은 우선 $n > 1$임을 확인하고, `A[0]`, $\cdots$, `A[n]` 중 최댓값을 찾아 `A[last]`와 교환한다. 이후 `A[0 : n-1]`에 대하여 재귀적으로 selection sort를 호출한다. 이때 이는 크기가 n인 배열이므로, 정렬이 성공적으로 수행된다. 0번째부터 $(n-1)$번째 원소까지 성공적으로 정렬되어있고, n번째 자리에 배열 전체의 최댓값이 위치해 있으므로, 크기가 $(n+1)$인 이 경우에도 정렬은 성공적으로 된다고 할 수 있다. 따라서 수학적 귀납법에 의하여 selection sort 알고리즘은 언제나 성공적인 sorting을 한다고 할 수 있다.

5번

Quicksort의 worst case는 pivot이 항상 최댓값 혹은 최솟값으로 선택되는 경우에 발생한다. Pivot을 중간값으로 선택하는 경우 원하는 정렬을 할 수 있다. 아래 알고리즘을 통하여 $\Theta(n)$의 시간 안에 크기가 n인 배열의 중간값의 인덱스을 찾을 수 있다.

정확히는 배열의 원소 중 i번째로 작은 원소를 찾는 것인데, 배열을 크기가 5인 배열들로 나누어 각각의 중간값을 구한 뒤, 그 값들의 배열을 다시 크기 5인 배열들로 나누어 각각의 중간값을 구하는 과정을 반복하여 적절한 pivot을 찾는다. 이 pivot은 전체 배열의 중간값에 가까운 값으로 선택될 것이다. 이를 기준으로 배열을 partition한 뒤, i 번째 원소가 포함되어 있는 쪽을 선택하여 이 전체 과정을 재귀적으로 반복하여 원하는 값을 찾을 수 있다.

Algorithm 1 5번 - 1

procedure FINDITHINDEX($A[]$, p, r, i) ▷ Find the index of the ith smallest element in $A[p, \cdots, r]$
  if $r - p + 1 < 5$ **then**
    return median index of A
  end if
  $M[] \leftarrow \Big[A\Big[\text{FINDITHINDEX}(A,\ p,\ p+4,\ 3)\Big],\ \cdots,\ A\Big[\text{FINDITHINDEX}(A,\ r-4,\ r,\ 3)\Big]\Big]$
  $pivotIndex \leftarrow \text{FINDITHINDEX}\left(M,\ 1,\ \frac{r-p+1}{5},\ \frac{r-p+1}{10}\right)$
  $q \leftarrow \text{PARTITION}(A,\ p,\ r,\ pivotIndex)$
  if $q - p \geq i$ **then**
    return FINDITHINDEX(A, p, $q-1$, i)
  else
    return FINDITHINDEX(A, q, r, $i - q + p$)
  end if
end procedure
procedure PARTITION($A[]$, p, r, $pivotIndex$) ▷ Relocate elements and return pivot index
  Relocate elements of $A[p, \cdots, r]$ using $A[pivotIndex]$ as the pivot.
  return final index of the pivot
end procedure

전체 배열의 크기를 n이라 하자. 크기가 5인 배열의 중간값을 구하는 과정은 각각 $\Theta(1)$의 시간을 소요하고, 총 $\frac{n}{5}$번 반복하므로 $\Theta(n)$의 시간이 소요된다. 이를 대상으로 다시 중간값을 구하는 과정에는 `FindIthIndex()`가 다시 호출되므로 $T\left(\frac{n}{5}\right)$의 시간이 소요된다. `Partition()`은 배열의 전 원소를 pivot과 한번씩 비교하므로 실행시간이 $\Theta(n)$이다. 이때 선택된 pivot은 중간값들의 중간값이므로, worst case에서 분할을 $\frac{3n}{10}$, $\frac{7n}{10}$의 크기로 하고, 재귀 단계에서 최대 $T\left(\frac{7n}{10}\right)$의 시간을 소요한다. 크기가 n인 배열에 대하여 `FindIthIndex()`의 전체 실행 시간을 $T_I(n)$이라 하면, 복잡도는 다음과 같다. 편의상 $n = 5^k$라 가정한다.

$$T_I(n) \leq \Theta(n) + T_I\left(\frac{n}{5}\right) + \Theta(n) + T_I\left(\frac{7n}{10}\right) = T_I\left(\frac{n}{5}\right) + T_I\left(\frac{7n}{10}\right) + \Theta(n)$$

어떤 양의 상수 c가 존재하여 $k < n$인 모든 k에 대하여 $T_I(k) \leq ck$을 만족한다고 가정하자. 그러면 충분히 큰 n에 대하여 다음이 성립한다.

$$\begin{aligned} T_I(n) &\leq T_I\left(\frac{n}{5}\right) + T_I\left(\frac{7n}{10}\right) + \Theta(n) \\ &\leq \frac{cn}{5} + \frac{7cn}{10} + \Theta(n) = cn - \frac{cn}{10} + \Theta(n) \\ &\leq cn - \frac{cn}{10} + dn\ (\exists d > 0) \\ &\leq cn \end{aligned}$$

$c \geq 10d$로 선택해주면 $-\dfrac{cn}{10} + dn < 0$이 되어 위 관계가 성립한다. 따라서 충분히 큰 모든 정수 n에 대하여 $T(n) \leq cn$이 성립한다고 할 수 있다. 따라서 $T_I(n) = O(n)$.

따라서 전체 quicksort은 다음과 같이 개선할 수 있다.

Algorithm 2 5번 - 2

procedure EnhancedQuicksort($A[]$, p, r) ▷ Sort $A[p, \cdots, r]$
  if $p < r$ **then**
    $pivotIndex \leftarrow$ FindIthIndex$\left(A,\ p,\ r,\ \left\lceil \dfrac{r-p+1}{2} \right\rceil\right)$
    $q \leftarrow$ Partition(A, p, r, $pivotIndex$)
    EnhancedQuicksort(A, p, $q-1$)
    EnhancedQuicksort(A, $q+1$, r)
  end if
end procedure

매 sorting 과정마다 pivot이 중간값으로 선택된다. 크기가 n인 배열에 대하여 이 알고리즘의 실행 시간을 $T(n)$이라 한다면 다음과 같이 계산된다. 편의상 $n = 2^k$라 가정한다.

$$\begin{aligned}
T(n) &= O(n) + \Theta(n) + 2T\left(\frac{n}{2}\right) = 2O\left(2^k\right) + 2T\left(2^{k-1}\right) \\
&= 2O\left(2^k\right) + 2\left(2O\left(2^{k-1}\right) + 2T\left(2^{k-2}\right)\right) = 2O\left(2^k\right) + 2^2O\left(2^{k-1}\right) + 2^2T\left(2^{k-2}\right) \\
&= \cdots = 2O\left(2^k\right) + \cdots + 2^{k-1}O\left(2^2\right) + 2^{k-1}T(2) \\
&= (k-1)O\left(2^{k+1}\right) + O\left(2^k\right) \\
&= O\left(k2^k\right) = O(n \log n)
\end{aligned}$$

따라서 총 알고리즘의 시간복잡도가 $O(n \log n)$으로, worst case에서도 유지된다.

6번

Algorithm 3 6번

procedure ENHANCEDMERGESORT($A[]$, p, r) ▷ Sort $A[p, \cdots, r]$
  if $p < r$ **then**
    $q_1, q_2, \cdots, q_{15} \leftarrow \lfloor \frac{15p+r}{16} \rfloor, \lfloor \frac{14p+2r}{16} \rfloor, \cdots, \lfloor \frac{p+15r}{16} \rfloor$
    ENHANCEDMERGESORT(A, p, q_1)
    ENHANCEDMERGESORT(A, q_1+1, q_2)
      $\vdots$
    ENHANCEDMERGESORT(A, $q_{15} + 1$, r)
    MERGE(A, p, q_1, q_2, $\cdots$, q_{15}, r)
  end if
end procedure
procedure MERGE($A[]$, p, q_1, q_2, $\cdots$, q_{15}, r) ▷ Merge 16 sorted arrays
  $o \leftarrow$ empty array of size $r - p + 1$
  $h \leftarrow$ min-heap made of the first elements of $A[p, \cdots, q_1]$, $A[q_1 + 1, \cdots, q_2]$, $\cdots$, $A[q_{15} + 1, \cdots, r]$
  while h is not empty **do**
    Delete the minimum element of h and insert it to o.
    if the array from which the deleted element came from is not empty **then**
      Insert the next element of the array to h.
    end if
  end while
end procedure

배열의 길이가 n일 때 실행시간을 $T(n)$이라 하자. 이때 $T(1) = 1$이다.

`Merge()`는 크기가 16인 힙을 만드는 과정, 힙에 $(n-16)$개의 원소를 추가하는 과정, 그리고 최상위 원소를 n번 순차적으로 삭제하는 과정을 포함한다. 이때 일반적으로 크기가 m인 힙에서 `buildHeap()`의 실행시간은 $\Theta(m)$, `insert()`의 실행시간은 $O(\log m)$, `deleteMin()`의 실행시간은 $O(\log m)$이다. 매 순간 힙의 크기는 16을 넘지 않는다. 따라서 총 실행시간은 힙 만들기에 $\Theta(16) = \Theta(1)$, 원소 추가에 최대 $O(\log 16) \times (n-16) = O(n)$, 최솟값 삭제에 최대 $O(\log 16) \times n = O(n)$이고, 총 `Merge()` 실행시간은 $O(n)$라 할 수 있다.

따라서 $n = 16^k$이라 가정할 때, 전체 실행시간은 다음과 같다.

$$\begin{aligned} T(n) &= 16T\Big(\frac{n}{16}\Big) + O(n) \\ &= 16\Big(16T\Big(\frac{n}{16^2}\Big) + O\Big(\frac{n}{16}\Big)\Big) + O(n) = 16^2 T\Big(\frac{n}{16^2}\Big) + 2O(n) \\ &= 16^3 T\Big(\frac{n}{16^3}\Big) + 3O(n) = \cdots = 16^k T(1) + kO(n) \\ &= n + O(n \log n) = O(n \log n) \end{aligned}$$

7번

(1) 한 번의 `mergeSort()` 호출에서 재귀 부분을 제외하고 한 번씩 `merge()`가 호출되므로, 하나의 `tmp[]` 배열이 생성된다. 배열의 분할은 크기가 1인 배열이 될 때까지 이루어지므로, $n = 2^k$라고 가정하면 `merge()`의 총 호출 횟수는 $1 + 2 + 2^2 + \cdots + 2^k = 2^{k+1} - 1 = 2n - 1$회이다. 따라서 `tmp[]` 배열 역시 약 $2n - 1 = \Theta(n)$개 생성된다.

(2) 각 `merge()` 단계에서 생성되는 `tmp[]` 배열의 크기는 $(r - p + 1)$이다. $n = 2^k$라고 가정하면, 호출되는 `merge()`에서 배열의 크기는 2^k인 경우 1번, 2^{k-1}인 경우 2번, $\cdots$, 1인 경우 2^k번 등장한다. 따라서 모든 `tmp[]` 배열들의 크기의 합은 다음과 같다.

$$\begin{aligned}\mathrm{Sum} &= 2^k \times 1 + 2^{k-1} \times 2 + \cdots + 1 \times 2^k \\ &= (k+1)2^k = \\ &= \Theta\big(k2^k\big) = \Theta(n \log n)\end{aligned}$$