EE608, Homework #1 Solution

(1) CLR 1.2-2

INSERTION-SORT beats MERGE-SORT when $8n^2 < 64n \lg n$, $\Rightarrow n < 8 \lg n$, $\Rightarrow \frac{n}{8} < \lg n$, $\Rightarrow 2^{n/8} < n$, which is true when $2 \leq n \leq 43$.

(You can solve for n by trial and error using a calculator. Observe that $5 \cdot 8 < n < 6 \cdot 8 = 48$, since $2^5 = 32 < 40$ and $2^6 = 64 > 48$, then narrow it down with the calculator.)

(2) CLR 1.2-3

ALGORITHM 1 will be faster when $100n^2 < 2^n$, $\Rightarrow n \geq 15$.

(You can solve for n by trial and error using a calculator.)

(3) CLR 1.1

	1 second	1 minute	1 hour	1 day
$\lg n$	2^{10^6}	$2^{6\times10^7}$	$2^{3.6\times10^9}$	$2^{8.64\times10^{10}}$
$\sqrt{n}$	10^{12}	3.6×10^{15}	1.269×10^{19}	7.4649×10^{21}
n	10^6	6×10^7	3.6×10^9	8.64×10^{10}
$n \lg n$	6.2746×10^4	2.801417×10^6	1.33378058×10^8	2.755147513×10^9
n^2	10^3	7.745×10^3	6×10^4	2.93938×10^5
n^3	100	391	1532	4420
2^n	19	25	31	36
$n!$	9	11	12	13

	1 month	1 year	1 century
$\lg n$	$2^{2.59\times10^{12}}$	$2^{3.1536\times10^{13}}$	$2^{3.1556736\times10^{15}}$
$\sqrt{n}$	6.7081×10^{24}	$9.94519296 \times 10^{26}$	$9.95827587 \times 10^{30}$
n	2.59×10^{12}	3.1536×10^{13}	3.1556736×10^{15}
$n \lg n$	$7.1870856404 \times 10^{10}$	$7.97633893349 \times 10^{11}$	$6.8654697441062 \times 10^{13}$
n^2	1.609347×10^6	5.615692×10^6	5.6175382×10^7
n^3	13733	31593	146677
2^n	41	44	51
$n!$	15	16	17

(4) CLR 2.1-1

The operation of INSERTION-SORT can be illustrated by Figure 1

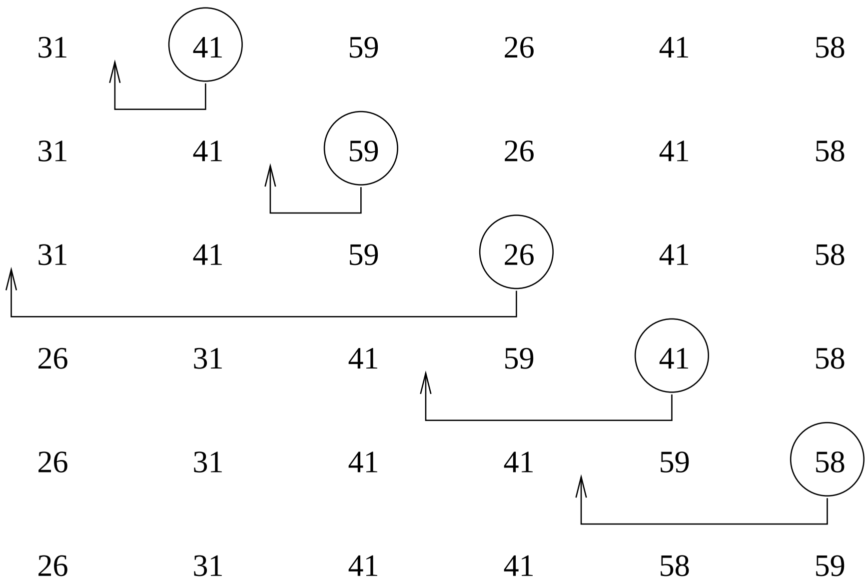

Figure 1: The operation of insertion sort for Problem CLR 2.1-1.

(5) CLR 2.1-2

```
NON-INCREASING-INSERTION-SORT(A)
for j ← 2 to length[A]
    do key ← A[j]
        ▷ Insert A[j] into the sorted sequence A[1..j − 1]
        i ← j − 1
        while i > 0 and A[i] < key
            do A[i + 1] ← A[i]
               i ← i − 1
        A[i + 1] ← key
```

(6) CLR 2.1-3

The pseudocode is as follows:

```
LINEAR-SEARCH(A, v)
j ← 1
while j ≤ length[A]
      if A[j] = v
         then return j
         else j ← j + 1
return NIL
```

We need to show the algorithm is correct by using a loop invariant in the algorithm. The loop invariant is as follows:

At the start of the jth loop iteration of the while loop in lines 2-5, the subarray $A[1..j-1]$ consists of elements not equal to v.

Now we will show that the loop invariant satisfies the three properties mentioned in the book:

Initialization: When $j = 1$, $A[1..j-1]$ contains no elements; hence, there can be no elements in $A[1..j-1]$ that are equal to v. Thus, the loop invariant holds.

Maintenance: If at the start of the $j = i$ iteration, the subarray $A[1..i-1]$ contains no elements equal to v, then there are two conditions to check. If $A[j] = v$, then we exit the loop and j is not incremented; hence, the loop invariant still holds. If $A[j] \neq v$, then j is incremented to $i+1$ and $A[1..i]$ contains no elements equal to v at the start of the $j = i+1$ iteration; hence, the loop invariant still holds.

Termination: If the loop terminates with $j = \mathit{length}[A] + 1$, then $A[1..\mathit{length}[A]]$ contains no elements equal to v, and the loop invariant holds. The algorithm will then return NIL at line 6, which is the correct response for not finding an element in A equal to v. If the loop terminates with $j < \mathit{length}[A] + 1$, then the algorithm has found an element $A[j]$ which is equal to v, returning key j at line 4 without incrementing j. Note that it is still the case that no elements in $A[1..j-1]$ are equal to v, and so the loop invariant holds.

Hence, the algorithm is correct when there is an element in A which is equal to v and when there is no such element.

(7) CLR 2.2-2

SELECTION-SORT(A)
$n \leftarrow \mathit{length}[A]$
for $i \leftarrow 1$ to $n-1$
  do $\mathit{min} \leftarrow \infty$
    $\triangleright$ Find smallest element of current portion of $A[i..n]$
    for $j \leftarrow i$ to n
      if $A[j] < \mathit{min}$
        do $\mathit{min} \leftarrow A[j]$
          $\mathit{key} \leftarrow j$
    $\triangleright$ Swap the smallest $A[\mathit{key}]$ with the element $A[i]$ and work with $A[i+1...n]$
    exchange $A[i] \leftrightarrow A[\mathit{key}]$
return A

The worst case running time for SELECTION-SORT is $\Theta(n^2)$, because the algorithm must find the next smallest element for each iteration. The best case could be $O(n)$ if the array is sorted by adding a linear check to see if the array is already sorted; however, without this check the best case is $\Theta(n^2)$.

The loop invariant is that at the start of the ith iteration $A[1..i-1]$ contains the smallest $i-1$ elements of A in sorted order.

At the end of the $(n-1)$th iteration, the nth element of A is no smaller than the first $n-1$ elements (by the loop invariant). And since $A[1..n-1]$ is sorted at this point, the whole array is sorted. Thus, there is no need for the nth iteration.

(8) CLR 2.2-4

Just modify the algorithm to store a precomputed answer for some of the inputs. Check for those inputs at the outset, and return the corresponding answer if it is detected.

(9) CLR 2.3-3

Base Case: For $n = 2$, $\mathrm{T}(2) = 2\lg 2 = 2$

Induction Hypothesis: Assume for $n = 2^k$,$\mathrm{T}(2^k) = 2^k \lg 2^k$.

Induction Step: Show this is true for $n = 2^{k+1}$, that is, $\mathrm{T}(2^{k+1}) = 2^{k+1} \lg 2^{k+1}$.

$$
\begin{aligned}
\mathrm{T}(2^{k+1}) &= 2\mathrm{T}(\frac{2^{k+1}}{2}) + 2^{k+1} \\
&= 2\mathrm{T}(2^k) + 2^{k+1} = 2(2^k \lg 2^k) + 2^{k+1} \\
&= 2^{k+1}(\lg 2^k + 1) = 2^{k+1}(\lg 2^k + \lg 2) \\
&= 2^{k+1}(\lg(2 \times 2^k)) = 2^{k+1}\lg(2^{k+1})
\end{aligned}
$$

(10) CLR 2.3-4

The running time for the recursive version of INSERTION-SORT is the following:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1, \\ T(n-1) + C(n) & \text{otherwise.} \end{cases}$$

where $C(n)$ is the time to insert the n^{th} element into $A[1...n-1]$.

(11) CLR 2.3-7

FindMatch(S,x) ;return true if found, false if not

{

1. Mergesort(S,1,n) ; $\Theta(n \cdot \log n)$

2. if $x > S[n] + S[n-1]$ or $x < S[1] + S[2]$ return false ; this line is optional, it can save some time by skipping the search for cases where x is too small or too large to be the sum of a pair of values in the sequence.

```
for i ← 1 to n
   do if (BinarySearch(S, x − S[i])=true); Θ(log n)
        then return true
return false
}
```

(12) CLR 2-2

(a) The output $A'[1]...A'[n]$ should be a permutation of the input $A[1]...A[n]$.

(b) The loop-invariant for the loop in lines 2-4 is that at the end of each iteration $A[j-1] \leq A[k]$ for all $k > j-1$. This can be proven as follows:

Initialization: Before the first iteration either $A[n-1] \leq A[n]$ or $A[n-1] > A[n]$. In the former case, the invariant will be true at the end of the first iteration because the **If** in line 3 evaluates to false and the exchange does not take place. In the latter case the **If** in line 3 evaluates to true, causing the last elements of the array A to be exchanged, and ensuring the invariant to hold true at the end of the iteration.

Maintenence: At the beginning of each iteration, $A[j]$ is known to be no greater than $A[k]$ for all $k > j$. During each iteration the elements $A[j]$ and $A[j-1]$ are compared and possibly exchanged such that the greater of the two is placed in $A[j]$. This ensures that at the end of the iteration $A[j-1]$ is not greater than $A[j]$, and by transitivity also not greater than $A[k]$ for all $k > j-1$.

Termination: At termination $j = i+1$, so $j-1 = i$. Because of the loop invariant we have that $A[i] \leq A[k]$ for all $k > i$.

(c) The loop-invariant for the loop in lines 1-4 is that at the end of each iteration the $A[1] \leq A[2] \leq A[3] ... \leq A[i]$.

Initialization: At the end of the first iteration of the outer loop $i = 1$ and the loop-invariant is trivially true.

Maintenence: The termination condition of the inner loop as proved in part(b) ensures that at the end of each iteration of the outer loop $A[i] \leq A[k]$ for all $k > i$. In the next iteration the element that is brought into $A[i+1]$ will be chosen from amongst $A[k]$ for $k > i$. We are guaranteed from the termination condition of the inner loop in the previous iteration of the outer loop that this element $A[i+1] \geq A[i]$. Hence on each iteration of the outer loop the element brought into $A[i]$ is such that $A[i-1] \leq A[i]$. The elements $A[1]...A[i-1]$ are not disturbed during the i'th iteration. So at the end of the i'th iteration we have that $A[1] \leq A[2] \leq A[3] ... \leq A[i]$.

Termination: At the end of the last iteration of the outer loop $i = length[A]$ and the loop-invariant ensures that $A[1] \leq A[2] \leq A[3] ... \leq A[i]$. Thus the algorithm terminates with the array A sorted in non-ascending order.

(d) The outer loop iterates $n = length[A]$ times. The inner loop iterates i times where i increases by 1 on each iteration of the outer loop. The running time of the algorithm can be summed as:

$$T(n) = \sum_{i=1}^{n}\sum_{j=i}^{n} c_1$$

$$T(n) = \sum_{i=1}^{n}(n-i)c_1$$

$$T(n) = \sum_{i=1}^{n} nc_1 - \sum_{i=1}^{n} ic_1$$

$$T(n) = n^2c_1 - \frac{n(n+1)}{2}c_1$$

$$T(n) = \frac{1}{2}n^2c_1 - \frac{n}{2}c_1$$

$$T(n) = \Theta(n^2)$$

The running time for BUBBLESORT is the same as the worst-case running time for INSERTION-SORT

(13) CLR 2-4

(a) The input sequence, $\langle 2, 3, 8, 6, 1\rangle$, has the following inversions: $\{(1, 5), (2, 5), (3, 5), (4, 5), (3, 4)\}$.

(b) The array with the most inversions contains unique elements and is reverse sorted, for example, $A = \{n, n-1, \ldots, 1\}$. In this case, there are the following number of inversions:

$$N_{inv} = (n-1) + (n-2) + \cdots + (n-(n-1)) = \sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

(c) Each element j of the array that is not in the correct position is shifted left until it reaches the element i such that $A[i] > j$. The number of shifts performed by insertion sort corresponds to the number of inversions of every element k, $j \leq k < i$ in the relation to i (i.e., inversion(k,i)). Each shift eliminates an inversion from the set.

(d) The algorithm to determine the number of inversions in $\Theta(n \lg n)$ uses MERGE-SORT with an enhancement in the MERGE procedure. When an array A_{low} (A with the lower index range) is merged with an array A_{high} (A with the greater index range, further from the index 1), if an element is picked from the top of A_{high}, the number of elements left in A_{low} should be added to form the total number of inversions at the end of the algorithm.

For example, $\langle 5, 2, 4, 6, 1, 3, 2, 6\rangle$ has $N_{inv} = 13$, using the MERGE-SORT enhancement (see Figure 2).

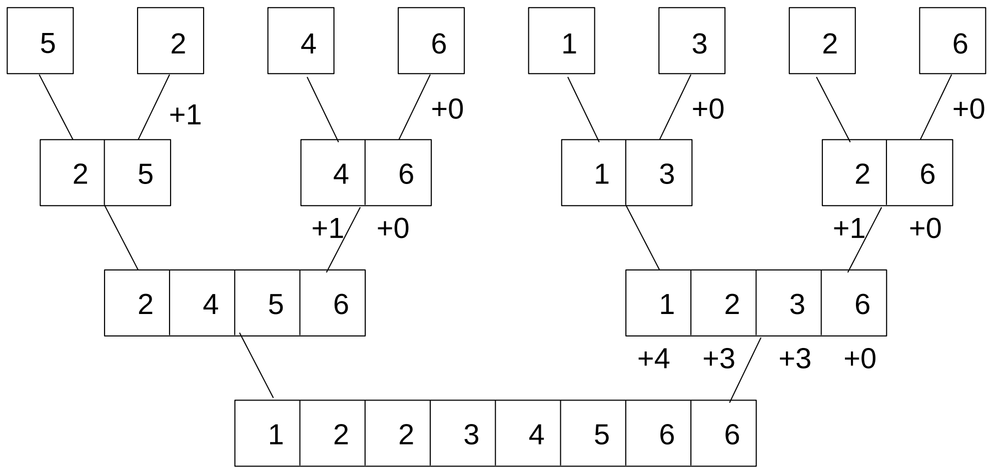

Figure 2: The MERGE-SORT enhancement for the example in Problem CLR 2-4(d).

MERGE-SORT can be modified to count inversions by adding line 18 to MERGE, as shown below. Note that *Num_of_inv* is a global variable initialized to zero that contains the total number of inversions when MERGE-SORT terminates. This will not change the asymptotic behavior of MERGE-SORT.

MERGE(A, p, q, r)
1. $n_1 \leftarrow q - p + 1$
2. $n_2 \leftarrow r - q$
3. Create arrays $L[1..n_1 + 1]$ and $R[1..n_2 + 1]$
4. **for** $i \leftarrow 1$ **to** n_1
5.   **do** $L[i] \leftarrow A[p + i - 1]$
6. **for** $j \leftarrow 1$ **to** n_2
7.   **do** $R[j] \leftarrow A[q + j]$
8. $L[n_1 + 1] \leftarrow \infty$
9. $L[n_2 + 1] \leftarrow \infty$
10. $i \leftarrow 1$
11. $j \leftarrow 1$
12. **for** $k \leftarrow p$ **to** r
13.   **do if** $L[i] \leq R[j]$
14.   **then** $A[k] \leftarrow L[i]$
15. $i \leftarrow i + 1$
16.   **else** $A[k] \leftarrow R[j]$
17. $j \leftarrow j + 1$
18. $Num\_of\_inv \leftarrow Num\_of\_inv + n_1 + 1 - i$

(14) CLR A.1-1

$$\sum_{k=1}^{n}(2k-1) = 2\sum_{k=1}^{n} k - \sum_{k=1}^{n} 1 = \frac{2n(n+1)}{2} - n = n(n+1) - n = n^2$$

(15) CLR A.1-6

Prove that $\sum_{k=1}^{N} O(f_k(n)) = O(\sum_{k=1}^{N} f_k(n))$.

Proof

Let $A = \sum_{k=1}^{N} O(f_k(n))$, and $B = O(\sum_{k=1}^{N} f_k(n))$.
Because A and B are sets of functions, we have to show $A \subseteq B$ and $B \subseteq A$.

(1) $A \subseteq B$

Let $g_k(n) = O(f_k(n))$ for $k \in \{1, 2, \ldots, n\}$. Then, there exist $c_k > 0$ and $n_k > 0$ such that

$$0 \leq g_k(n) \leq c_k f_k(n) \qquad \forall n \geq n_k$$

Choose $c = \max_{k \in \{1,2,\ldots,n\}}\{c_k\}$ and $n_0 = \max_{k \in \{1,2,\ldots,n\}}\{n_k\}$. Then, we have

$$0 \leq g_k(n) \leq c_k f_k(n) \leq c f_k(n) \qquad \forall n \geq n_0$$

By the linear property of the summation,

$$0 \leq \sum_{k=1}^{N} g_k(n) \leq c \sum_{k=1}^{N} f_k(n) \qquad \forall n \geq n_0$$

Hence, $\sum_{k=1}^{N} g_k(n) = O(\sum_{k=1}^{N} f_k(n))$.
Because the above equation holds for an arbitrary $g_k(n)$, we have:

$$\sum_{k=1}^{N} O(f_k(n)) \subseteq O(\sum_{k=1}^{N} f_k(n))$$

.

(2) $(B \subseteq A)$

Let $g(n) = O(\sum_{k=1}^{N} f_k(n))$. Then, there exist $c > 0$ and $n_0 > 0$ such that

$$0 \leq g(n) \leq c \sum_{k=1}^{N} f_k(n) \qquad \forall n \geq n_0$$

By the linear property of summations,

$$0 \leq g(n) \leq \sum_{k=1}^{N} [c f_k(n)] \qquad \forall n \geq n_0$$

If we choose $c_k = c$ and $n_k = n_0$ for $k \in \{1, 2, \cdots, n\}$ in the above inequality, we obtain that there exist $c_k > 0$ and $n_k > 0$ for $k \in \{1, 2, \cdots, n\}$, such that $0 \leq g(n) \leq c_1 f_1(n) + c_2 f_2(n) + \cdots + c_n f_n(n)$ whenever $n \geq n_1, n \geq n_2, \cdots, n \geq n_n$.
This means that $g(n) = \sum_{k=1}^{N} O(f_k(n))$.
Therefore, $O(\sum_{k=1}^{N} f_k(n)) \subseteq \sum_{k=1}^{N} O(f_k(n))$.

From (1) and (2), we have $A = B$.

(16) CLR A.1-7

One method of solution uses a logarithm to convert the product to a sum:

$$\begin{aligned}
\prod_{k=1}^{n} 2 \cdot 4^k &= 2^n \prod_{k=1}^{n} 4^k \\
\lg(2^n \prod_{k=1}^{n} 4^k) &= \lg(2^n) + \lg(\prod_{k=1}^{n} 4^k) \\
&= \lg(2^n) + \sum_{k=1}^{n} \lg(4^k) \\
&= \lg(2^n) + \lg(4) \sum_{k=1}^{n} k \\
&= \lg(2^n) + 2\lg(2)\frac{n(n+1)}{2} \\
&= n\lg(2) + n(n+1)\lg(2) \\
&= n(n+2)\lg(2) \\
&= \lg(2^{n(n+2)}) \\
\\
\prod_{k=1}^{n} 2 \cdot 4^k &= 2^{n(n+2)}.
\end{aligned}$$

An alternative method of solution uses product and exponent rules:

$$\begin{aligned}
\prod_{k=1}^{n} 2 \cdot 4^k &= 2^n \cdot 4^{\sum_{k=1}^{n} k} \\
&= 2^n \cdot 4^{\frac{n(n+1)}{2}} \\
&= 2^n \cdot 2^{n(n+1)} \\
&= 2^{n+n(n+1)} \\
&= 2^{n^2+2n} \\
&= 2^{n(n+2)}
\end{aligned}$$

(17) CLR A.2-4

Evaluate $\sum_{k=1}^{n} k^3$

Since k^3 is a monotonically increasing function,

$$\int_0^n x^3 dx \leq \sum_{k=1}^{n} k^3 \leq \int_1^{n+1} x^3 dx \leq \int_0^{n+1} x^3 dx$$

Hence:

$$\frac{x^4}{4}\Big|_0^n \leq \sum_{k=1}^{n} k^3 \leq \frac{x^4}{4}\Big|_0^{n+1}$$

Giving:

$$\tfrac{n^4}{4} \leq \sum_{k=1}^{n} k^3 \leq \tfrac{(n+1)^4}{4}$$

Hence, $\sum\limits_{k=1}^{n} k^3 = \Theta(n^4)$

(18) CLR A.2-5

If you bound the summation using an integral directly (the integral bounds are from 0 to n) the final solutions becomes $[lnn - ln0]$.

However, $ln0$ is negative infinity.

So instead we break up the summation

$\sum\limits_{k=1}^{n} 1/k$ and rewrite it as $\sum\limits_{k=2}^{n} 1/k$ $+1$.

(19) CLR A-1

(a) Use the integral method of bounding the summation:

$$\int_0^n x^r\, dx \leq \sum_{k=1}^{n} k^r \leq \int_1^{n+1} x^r\, dx$$

$$\frac{1}{r+1} x^{r+1}\Big|_0^n \leq \sum_{k=1}^{n} k^r \leq \frac{1}{r+1} x^{r+1}\Big|_1^{n+1}$$

$$\frac{n^{r+1}}{r+1} \leq \sum_{k=1}^{n} k^r \leq \frac{(n+1)^{r+1} - 1}{r+1} \leq \frac{(n+1)^{r+1}}{r+1}$$

Therefore,

$$\sum_{k=1}^{n} k^r = \Theta(n^{r+1})$$

(b) On one hand,
$\sum_{k=1}^{n} \lg^s k \leq \sum_{k=1}^{n} \lg^s n = n \lg^s n$.
Clearly, $\sum_{k=1}^{n} \lg^s k = O(n \lg^s n)$.

On the other hand,
$\sum_{k=1}^{n} \lg^s k = \sum_{k=1}^{\lfloor n/2 \rfloor} \lg^s k + \sum_{k=\lfloor n/2 \rfloor+1}^{n} \lg^s k$
$\geq \sum_{k=1}^{\lfloor n/2 \rfloor} 0 + \sum_{k=\lfloor n/2 \rfloor+1}^{n} \lg^s (n/2)$
$\geq (n/2) \lg^s (n/2)$.
We can easily prove that $(n/2) \lg^s (n/2) = \theta(n \lg^s n)$.

So $\sum_{k=1}^{n} \lg^s k = \theta(n \lg^s n)$.

(c)

On one hand,

$$\sum_{k=1}^{n} k^r lg^s k \leq \sum_{k=1}^{n} k^r lg^s n = lg^s n \sum_{k=1}^{n} k^r$$

and from part (a)

$$\sum_{k=1}^{n} k^r = \Theta(n^{r+1})$$

so,

$$\sum_{k=1}^{n} k^r lg^s k = O(n^{r+1} lg^s n)$$

On the other hand,

$$\sum_{k=1}^{n} k^r lg^s k = \sum_{k=1}^{n/2} k^r lg^s k + \sum_{k=n/2}^{n} k^r lg^s k$$

$$\sum_{k=1}^{n} k^r lg^s k \geq \sum_{k=1}^{n/2} 0 + \sum_{k=n/2}^{n} (n/2)^r lg^s (n/2)$$

$$\sum_{k=1}^{n} k^r lg^s k \geq (n/2)^{r+1} lg^s (n/2)$$

And then combining the above two results it can be shown that,

$$\sum_{k=1}^{n} k^r lg^s k = \Theta(n^{r+1} lg^s n)$$