Data Structure and Algorithm Analysis (H) Lab 01.

Q 1.1

iteration									
0:	24	5	6	23	42	45	2	1	8
1:	1	5	6	23	42	45	2	24	8
2:	1	2	6	23	42	45	5	24	8
3:	1	2	5	23	42	45	6	24	8
4:	1	2	5	6	42	45	23	24	8
5:	1	2	5	6	8	45	23	24	42
6:	1	2	5	6	8	23	45	24	42
7:	1	2	5	6	8	23	24	45	42
8:	1	2	5	6	8	23	24	42	45

Q 1.2

Proof:

* Loop Invariant: „At the ~~start~~ end of each iteration of the loop of lines 2-6, the subarray $A[1,\cdots,j]$ consists of the ~~elements originally in A[1,⋯,~~ smallest j elements in the whole array in sorted order."
* Initialization: For $j=1$, the subarray $A[1]$ becomes the array consisting of the smallest element in the array. ✓
* Maintenance: The ~~while~~ for loop ~~moves~~ swaps the smallest element in the subarray $A[j+1\cdots,n]$ with the element with index $j+1$ (in the $(j+1)$-th iteration). But it's still larger then those in $A[1,\cdots,j]$. so $A[1\cdots,j+1]$ is in sorted order:

$$\underbrace{A[1]\le A[2]\le\cdots\le A[j]}_{\text{sorted before}}\le\underbrace{A[j+1]}_{\text{from for loop}}\le\underbrace{A[j+2]\le\cdots\le A[n]}_{\text{still unsorted.}}$$

* Termination: The for loop ends when $j=n$. Then the loop invariant for $j=n$ says that the whole array contains the original $A[1\cdots n]$ in ascending (sorted) order.

Q 1.3

```
SELECTION-SORT(A).
n = A.length
for j = 1 to n-1 do.
        smallest = j.
        for i = j+1 to n do
                if A[i] < A[smallest] then smallest = i
        exchange A[j] with A[smallest].
```

\# of executions.

2

n

$n-1$

$n+(n-1)+\cdots+2 = \frac{n(n+1)}{2} - 1$

$\sum_{j=1}^{n-1} t_j$.

3.

Define t_j as the number of times the „[redacted] then" statement is executed for that j.

Since ~~the~~ cost for every execution is 1.

Q 1.3

SELECTION-SORT

Best case: ~~the~~ whole array is already sorted ~~before~~ we sort it.

running time: $1 + n\cdot 1 + (n-1)\cdot 1 + [n+(n-1)+\cdots+2]\cdot 1$

$+ [n + (n-1) + \cdots + 2]\cdot 1 + (n-1)\cdot 1$

$= 1 + n + n-1 + \frac{1}{2}n^2 + \frac{1}{2}n - 1 + \frac{1}{2}n^2 + \frac{1}{2}n - 1 + n - 1$

$= n^2 + 3n - 3 \triangleq an^2 + bn + c$. — quadratic.

Compared with [redacted] INSERTION-SORT: $T(n) = an+b$ — linear

$\Rightarrow$ INSERTION-SORT is much faster.

Worst Case: the whole

Q 1.3

⚠ I have to say that the hint has made some ambiguity. I stared at it for so long since if I just ~~data~~ ~~consider~~ ~~the~~ ignore the fact that the „then" may not be excuted, there isn't [redacted] „Best case" and „worst case" in the real sense since all the differences are made by then „then" statement.

[redacted] Well, I understand that this hint is to make clarification for such question: there would be no best /worst case.

So, let's do the problem by differentiating the difference and not differentiating it.

Case 1. If we don't specify the "then" statement.

Since the key difference between cases is that whether we need to swap the smallest one and the j-th element in our array.

So, for all cases:

$$T(n) = 1 + n + (n-1) + [(n-1)+\cdots+1+(n-1)] + [(n-1)+\cdots+1] + (n-1)$$

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

$$= n^2 + 3n - 2 \text{ — quadratic.}$$

Compared with Insertion Sort:

Best case: $T(n) = an+b$. — linear.

Worst case: $T(n) = an^2+bn+c$. — quadratic.

It seems that InsertionSort is more efficient.

Case 2: If we consider the "then" statement's effect.

Best case: If the array is already sorted.

$$T(n) = 1 + n + (n-1) + [n + (n-1) + \cdots + 2] + [n + (n-1) + \cdots + 2] + (n-1) = n^2 + 3n - 2 \text{ — quadratic.}$$

Compared with InsertionSort: $T(n) = an+b$ — linear — more efficient

Worst Case: If the array is in descending order.

$$T(n) = 1 + n + (n-1) + [(n-1) + \cdots + 1 + n - 1] + [(n-1) + \cdots + 1] \times 2 + (n-1)$$

$$= n^2 + 3n - 2 + \frac{1}{2}n^2 - \frac{1}{2}n = \frac{3}{2}n^2 + \frac{5}{2}n - 2 \text{ — quadratic.}$$

Compared with InsertionSort: $T(n) = an^2+bn+c$ — quadratic.

So, In general, InsertionSort is more efficient.