

1. Suppose Computer A is running a Sorting Algorithm and it is supposed to sort an array of ten million numbers. Suppose that Computer A executes a billion instructions per second, and suppose Computer A requires $100 n \lg n$ instructions to sort n numbers. Find the time it takes computer A to sort the ten million numbers?

Computer A

$$1 \text{ Million} = 10^6$$

$$10 \text{ Million} = 10^7$$

$$\text{Execute 1 Billion} = 10^9 \text{ instructions/second}$$

Requires $100 n \lg n$ instructions

$$\text{Sort 10 Million, } n = 10^7$$

$$\text{Time} = \frac{100 n \lg n \text{ instructions}}{10^9 \text{ instructions/second}}$$

$$= \frac{100 \times 10^7 \lg 10^7 \text{ inst}}{10^9 \text{ inst/second}}$$

$$= \lg 10^7 \text{ second}$$

$$= 7 \lg 10 \text{ second}$$

$$\text{i.e., } \lg n = \log_2 n$$

$$= 7 \log_2^{10} \text{ second}$$

$$= 7 \times 3.32 \text{ second}$$

$$= 23.24 \text{ second}$$

$$\text{Time} \approx 23.24 \text{ second}$$

2. Suppose we are comparing implementations of insertion sort and Merge Sort on the same machine. For inputs of size n , insertion sort runs in $2n^2$ steps, while merge sort runs in $10n \lg n$ steps. For which values of n does insertion sort beat merge sort?

For Insertion sort to beat merge sort for inputs of size n , $2n^2$ must be less than $10n \lg n$

$$2n^2 < 10n \lg n$$

$$n < 5 \lg n$$

$$\frac{n}{5} < \lg n$$

$$\frac{n}{5} < \log_2^n$$

$$2^{n/5} < n$$

Check for values of n which are power of 2.

$$n=5 \Rightarrow 2^{5/5}$$

$$n=10 \Rightarrow 2^{10/5} = 4 < n$$

$$n=20 \Rightarrow 2^{20/5} = 16 < n$$

$$n=25 \Rightarrow 2^{25/5} = 32 > n$$

$$n=21 \Rightarrow 2^{21/5} = 18.3 < n, \quad n=22 \Rightarrow 2^{22/5} = 21.1 < n$$

So, at $n=22$, Insertion sort starts to beat Merge Sort.

Therefore, for $2 \leq n \leq 22$, insertion sort beats Merge sort.

3. Using the example we went over in the class as a model, illustrate the operations of insertion-sort on the Array $A = \langle 3, 7, 5, 1, 8, 2 \rangle$

	1	2	3	4	5	6
A	3	7	5	1	8	2
	i	j				

j	Key	i
2	7	1
3	5	2
4	1	3
5	8	4
6	2	5
7	stop	

Steps:

1. $j=2$, $\text{Key}=7$, $i=j-1=2-1=1$

$i > 0$ and $A[i] > \text{Key}$

$1 > 0$ and $3 > 7$ // Fail

1	2	3	4	5	6
3	7	5	1	8	2
	i	j			

2. $j=3$, $\text{Key}=5$, $i=2$

$2 > 0$ and $7 > 5$, $A[i+1] = A[i]$, $A[3] = A[2]$

1	2	3	4	5	6
3	5	7	1	8	2
		i	j		

$i = i-1 = 2-1=1$

$1 > 0$ and $3 > 5$ // Fail

$A[i+1] = \text{Key}$, $A[2] = 5$

3. $j=4$, $\text{Key}=1$, $i=3$

$3 > 0$ and $7 > 1$, $A[4] = A[3]$

$i=2$

$2 > 0$ and $5 > 1$, $A[3] = A[2]$

$i=1$

$1 > 0$ and $3 > 1$, $A[2] = A[1]$

$i=0$

$A[i+1] = \text{Key}$, $A[1] = 1$

1	2	3	4	5	6
1	3	5	7	8	2

i j

4. $j=5$, Key=8, $i=4$

$4 > 0$ and $7 > 8$ // Fail

1	2	3	4	5	6
1	3	5	7	8	2

i j

5. $j=6$, Key=2, $i=5$

$5 > 0$ and $8 > 2$, $A[6] = A[5]$

$i=4$

$4 > 0$ and $7 > 2$, $A[5] = A[4]$

$i=3$

$3 > 0$ and $5 > 2$, $A[4] = A[3]$

$i=2$

$2 > 0$ and $3 > 2$, $A[3] = A[2]$

$i=1$

$1 > 0$ and $1 > 2$ // Fail

$A[i+1] = \text{Key}$, $A[2] = 2$

1	2	3	4	5	6
1	2	3	5	7	8

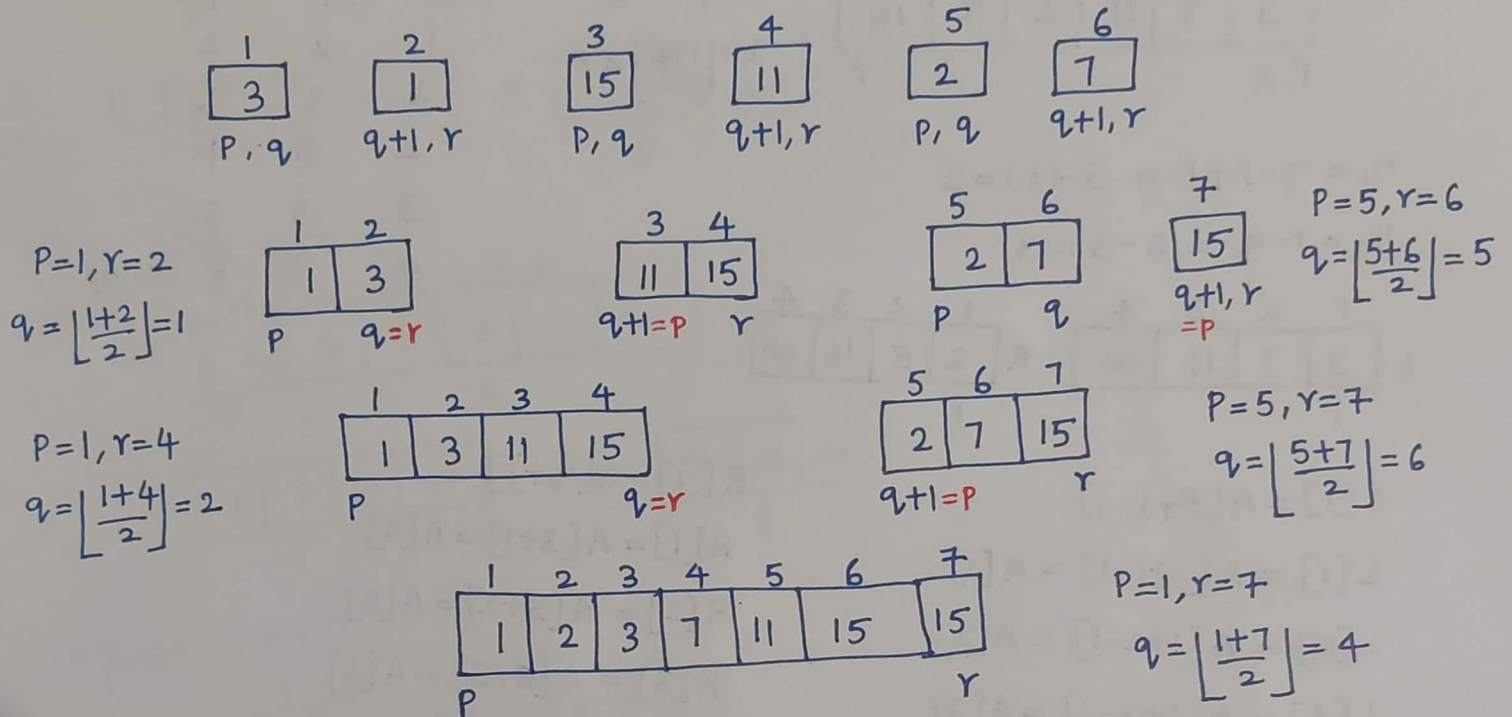
4. Rewrite the INSERTION-SORT procedure to sort into non increasing instead of non decreasing order.

INSERTION-SORT (A)

1. for $j = 2$ to $A.length$
2. Key = $A[j]$
3. // Insert $A[j]$ into the sorted sequence $A[1..j-1]$
4. $i = j - 1$
5. while $i > 0$ and $A[i] < Key$
6. $A[i+1] = A[i]$
7. $i = i - 1$
8. $A[i+1] = Key$

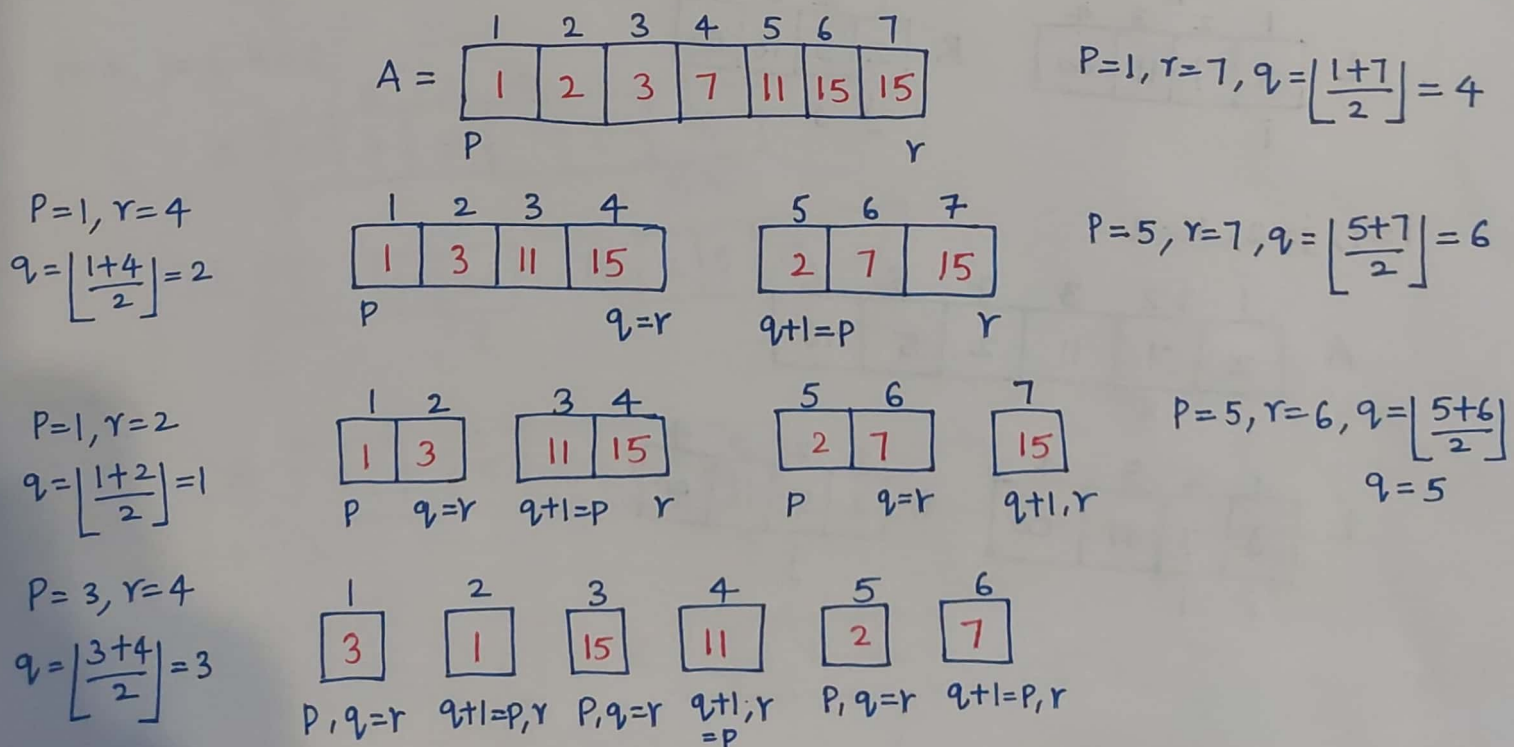
5. Use the top-down approach to illustrate the operations of merge-sort on the array $A = \langle 3, 1, 15, 11, 2, 7, 15 \rangle$. Use the notes discussed in class as a guide.

Let $A = \{3, 1, 15, 11, 2, 7, 15\}$



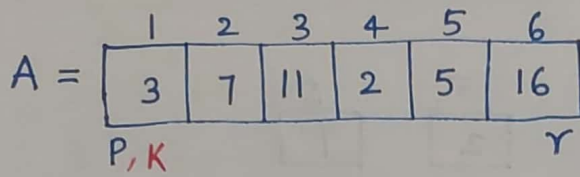
6. Use the bottom up approach to illustrate the operations of merge sort on the array $A = \langle 3, 1, 15, 11, 2, 7, 15 \rangle$. Use the notes discussed in class as guide.

$A = \{3, 1, 15, 11, 2, 7, 15\}$



7. Illustrate the Operations of merge on the Array $A = \langle 3, 7, 11, 2, 5, 16 \rangle$

Use the notes discussed in class as a guide.



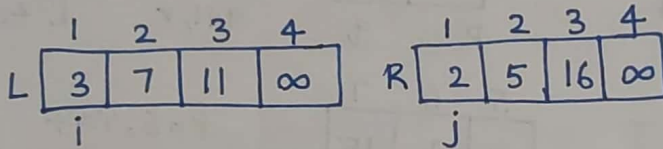
$$P=1, r=6$$

$$q = \left\lfloor \frac{1+6}{2} \right\rfloor = 3$$

i	j	K
1	1	1
1	2	2
2	2	3
2	3	4
3	3	5
4	3	6
		Stop

$$n_1 = q - P + 1 = 3 - 1 + 1 = 3$$

$$n_2 = r - q = 6 - 3 = 3$$



$$L[i] = A[P+i-1]$$

$$L[1] = A[1+1-1] = A[1]$$

$$L[2] = A[1+2-1] = A[2]$$

$$L[3] = A[1+3-1] = A[3]$$

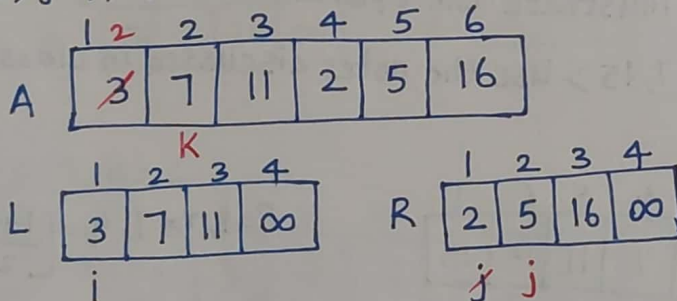
$$R[j] = A[q+j]$$

$$R[1] = A[3+1] = A[4]$$

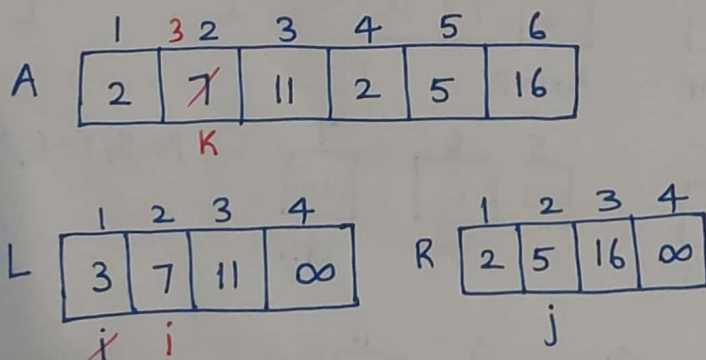
$$R[2] = A[3+2] = A[5]$$

$$R[3] = A[3+3] = A[6]$$

$$i=1, j=1, K=1$$



$$i=1, j=2, K=2$$



$$i=2, j=2, k=3$$

	1	2	3 ⁵	4	5	6
A	2	3	11	2	5	16
	p		K			r

L	3	7	11	∞		
			i			

R	2	5	16	∞		
		j	j			

$$i=2, j=3, k=4$$

	1	2	3	4 ⁷	5	6
A	2	3	5	2	5	16
	p			K		r

L	3	7	11	∞		
		j	i			

R	2	5	16	∞		
				j		

$$i=3, j=3, k=5$$

	1	2	3	4	5 ¹¹	6
A	2	3	5	7	5	16
	p				K	r

L	3	7	11	∞		
			j	i		

R	2	5	16	∞		
					j	

$$i=4, j=3, k=6$$

	1	2	3	4	5	6 ¹⁶
A	2	3	5	7	11	16
	p					r, K

L	3	7	11	∞		
				i		

R	2	5	16	∞		
			j	j		

6248

A	2	3	5	7	11	16
	p					r K

L	3	7	11	∞		
				i		

R	2	5	16	∞		
					j	

8. Rewrite the Merge procedure so that it does not use sentinels, instead stopping once either array L or R has had all its element copied back to A and then copying the remainder of the other array back into A.

Merge Algorithm Without Using Sentinels

Merge (A, p, q, r)

1. $n_1 = q - p + 1$
2. $n_2 = r - q$
3. Let $L[1 \dots n_1]$ and $R[1 \dots n_2]$ be new arrays.
4. for $i = 1$ to n_1
5. $L[i] = A[p + i - 1]$
6. for $j = 1$ to n_2
7. $R[j] = A[q + j]$
8. $i = 1$
9. $j = 1$
10. for $k = p$ to r
11. if $i > n_1$
12. $A[k] = R[j]$
13. $j = j + 1$
14. else if $j > n_2$
15. $A[k] = L[i]$
16. $i = i + 1$
17. else if $L[i] \leq R[j]$
18. $A[k] = L[i]$
19. $i = i + 1$
20. else
21. $A[k] = R[j]$
22. $j = j + 1$

9. Express the function $\frac{n^3}{1000} - 100n^2 - 100n + 3$ in terms of

θ notation.

The highest order of n term of the function ignoring the constant coefficient is n^3 . So, the function in θ -notation will be $\theta(n^3)$

$$\text{function} = \frac{n^3}{1000} - 100n^2 - 100n + 3$$

$$\theta(e) = \theta\left(\max\left(\frac{n^3}{1000} - 100n^2 - 100n + 3\right)\right)$$

$$\theta(e) = \theta(\max(n^3, n^2, n, 1))$$

$$\text{Now, } \theta(1) < \theta(n) < \theta(n^2) < \theta(n^3)$$

$$\text{So, } \theta(\max(n^3, n^2, n, 1)) = \theta(n^3)$$