Question 1

Given asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible, and justify your answers.

1. $T(n) = 2T(n/2) + n^4$**.**

2. $T(n) = T(7n/10) + n$**.**

3. $T(n) = 16T(n/4) + n^2$**.**

4. $T(n) = 7T(n/3) + n^2$**.**

5. $T(n) = 7T(n/2) + n^2$**.**

6. $T(n) = 2T(n/4) + \sqrt{n}$**.**

7. $T(n) = T(n-2) + n^2$**.**

1. According to master theory, $a = 2$, $b = 2$, $f(n) = n^4$. We have $f(n)/n^{log_b a} = n^4/n = n^3$. $af(n/b)/f(n) = 2f(n/2)/f(n) < 1$. It is case 3. So $T(n) = \Theta(n^4)$.

2. According to master theory, $a = 1$, $b = 10/7$, $f(n) = n$. We have $f(n)/n^{log_b a} = n/1 = n$. $af(n/b)/f(n) = 7/10 < 1$. It is case 3. So $T(n) = \Theta(n)$.

3. According to master theory, $a = 16$, $b = 4$, $f(n) = n^2$. We have $f(n)/n^{log_b a} = 1 = \Theta(lg^0 n)$. It is case 2. So $T(n) = \Theta(lgn)$.

4. According to master theory, $a = 7$, $b = 3$, $f(n) = n^2$. We have $f(n)/n^{log_b a} = n^{0.23} =$. $af(n/b)/f(n) = 7/9 < 1$. It is case 3. So $T(n) = \Theta(n^2)$.

5. According to master theory, $a = 7$, $b = 2$, $f(n) = n^2$. We have $f(n)/n^{log_b a} = n^{-0.81} =$. It is case 1. So $T(n) = \Theta(n^{log_2 7})$.

6. According to master theory, $a = 2$, $b = 4$, $f(n) = n^{\frac{1}{2}}$. We have $f(n)/n^{log_b a} = 1 = \Theta(lg^0 n)$. It is case 2. So $T(n) = \Theta(lgn)$.

7.

$$
\begin{aligned}
T(n) &= n^2 + T(n-2) \\
&= n^2 + (n-2)^2 + T(n-4) \\
&... \\
&= n^2 + (n-2)^2 + ... + 4^2 + 2^2 \text{或} = n^2 + (n-2)^2 + ... + 3^2 + 1^2
\end{aligned}
$$

We can get that following relations:

$$\begin{aligned} T(n) &\geq 1^2 + 3^3 + ... + (2\lceil\frac{n}{2}\rceil - 1)^2 \\ &> \frac{1}{2}(1^2 + 2^2 + 3^2 + ... + (2\lceil\frac{n}{2}\rceil - 1)^2) \\ &> \frac{1}{2}(1^2 + 2^2 + 3^2 + ... + (n-1)^2) \\ &= \frac{n(n-1)(2n-1)}{12} \\ &= \Theta(n^3) \end{aligned}$$

and

$$\begin{aligned} T(n) &\leq 2^2 + 4^3 + ... + (2\lceil\frac{n}{2}\rceil)^2 \\ &< \frac{1}{2}(2^2 + 3^2 + ... + (2\lceil\frac{n}{2}\rceil + 1)^2) \\ &< \frac{1}{2}(1^2 + 2^2 + 3^2 + ... + (2\frac{n+1}{2} + 1)^2) \\ &= \frac{1}{2}(1^2 + 2^2 + 3^2 + ... + (n+2)^2) \\ &< \frac{(n+2)(n+3)(2n+5)}{12} \\ &= \Theta(n^3) \end{aligned}$$

Therefore, we can conclude that $T(n) = \Theta(n^3)$

Question 2

Given an array $A =< 5, 13, 2, 25, 37, 17, 20, 8, 24, 9, 7 >$**, please show:**

1. A **after the BuildHeap(A) process;**
2. A **right after heap_size(A) turns into 8;**
3. A **right after heap_size(A) turns into 3.**

1. After the BuildHeap(A) process, the tree is showed in figure 2.1,and array $A =< 37, 25, 20, 24, 13, 17, 2, 8, 5, 9, 7 >$

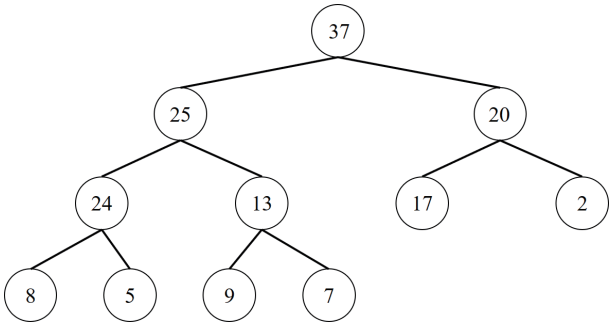

Figure 2.1 After the BuildHeap(A) process

2. After heap_size(A) turns into 8, the tree is showed in figure 2.2, and array $A =< 37, 25, 20, 8, 13, 17, 2, 5 >$

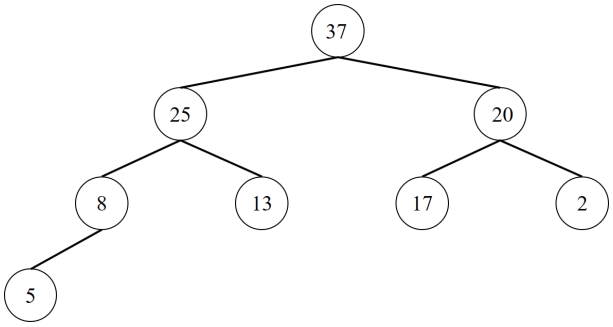

Figure 2.2 after heap_size(A) turns into 8

3. After heap_size(A) turns into 3, the tree is showed in figure 2.3, and array $A =< 13, 5, 2 >$

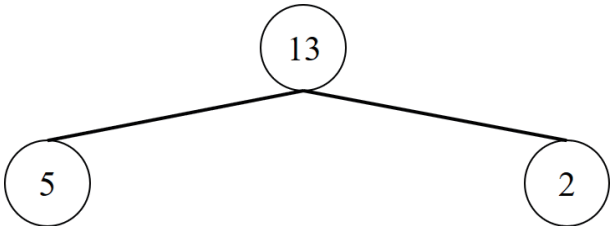

Figure 2.3 after heap_size(A) turns into 3

Question 3

What is the running time of HEAPSORT on an array A of length n that is already sorted in increasing order? What about decreasing order?

1. When n is already sorted in increasing order, each mid-node should be shift to the leaf lay. Then, it is obviously that the shift count of each node is its depth. We know that there are $n - \lfloor \frac{n}{2} \rfloor$ leaf nodes whose depth is 0. There are $\lfloor \frac{n}{2} \rfloor - \lfloor \frac{n}{4} \rfloor$ nodes whose depth is 1 ...

Therefore, we can get the running time $T(n) = \sum_{i=1}^{\lfloor logn \rfloor + 1} (\lfloor \frac{n}{2^{i-1}} \rfloor - \lfloor \frac{n}{2^i} \rfloor)(i-1)$.

2. When n is already sorted in decreasing order, we don't need to shift any nodes. Therefore the running time is $\lfloor \frac{n}{2} \rfloor$.

Question 4

Given an $O(nlgk)$-time algorithm to merge k sorted lists into one sorted list, where n is the total number of elements in all the input lists.(*Hint*: Use a min-heap for k-way merging).

We can generalize the 2-array merge to k-array merge. It is obviously that we can set k points and put each of them at the first elements of the k arrays. Then we should compare the k elements pointed by k the pointers and move the pointer pointing to the smallest one to the next element. As to the comparison of the k elements, we can use min-heap to store and sorted them. The algorithm is showed in Alg 4.0.1.

Algorithm 4.0.1 Merge k sorted array algorithm

Input: k sorted arrays $A[1], A[2], ..., A[k]$

```
1: Extract the first element of A[1], A[2], ..., A[k] respectively to get array B.
2: Build a min-heap H using array B
3: while H ≠ φ do
4:    Move the root r element to the result list R
5:    if the array A[j] contains r is not empty then
6:       Move the next element in A[j] to the root
7:       Heapify the heap
8:    else
9:       Move the last leaf to the root
10:      Heapify the heap
11:   end if
12: end while
13: Return R
```

We can see that it costs lgk time to get the minimal element in the current min-heap and the total number of elements is n. So the time of this merging is $O(nlgk)$.