Algorithm Assignment 2

> *10185101210 陈俊潼*

2.3-7

First using quick sort to sort the given array. The complexity is $\Theta(nlogn)$, then iterate the whole sorted array using the following algorithm:

```
IF_EXISTS_SUM(Array, sum):
    for i = 0 to Array.length:
        if (i > sum)
            continue
        else
            BINARY_SEARCH(sum - A, Array, i)

```

The complexity of binary search here is $\Theta(\log n)$, will have a total of n loops. The total complexity is $n\Theta(logn) + \Theta(n \log n) = \Theta(nlogn)$

4.1-5

```
MAX_SUBARRAY_LINEAR(A):
    currentBegin = 0
    currentEnd = 0
    currentSum = 0
    finalBegin = 0
    finalEnd = 0
    finalSum = 0
    for i = 0 to Array.length
        currentEnd = i
        if(currentSum > 0)
            currentSum = currentSum + A[i]
        else
            currentBegin = i
            currentSum = A[i]
        if currentSum > finalSum
            finamSum = currentSum
            finalBegin = currentBegin
            finalEnd = currentEnd
    return (finalBegin, finalEnd, finalSum)
```

6.3-3

The maximum node count for each layer of a heap is:

$$2^0 + 2^1 + 2^2 + \ldots + 2^h$$

Which $< 2^h + 2^h$, and 2^h is the maximum number of leaf nodes. So the maximum number of nodes for height 0 is $\lceil n/2 \rceil$. Then each higher layer for the heap has at most half of the nodes of the lower layer, so for height h, there are at most $\lceil n/2^{h+1} \rceil$ nodes.

4-1

a. From master theorem:

$$f(n) = n^4, a = 2, b = 2, n^{\log_b a} = n^1$$
$$\therefore \text{Complexity is } \Theta(n^4)$$

Validation:

$$\begin{aligned} T(n) = n^4 &\le 2cn^4/2^4 + n^4 \\ &= n^4 + cn^4/8, \forall c > 0 \text{ holds.} \end{aligned}$$

b. From master theorem:

$$f(n) = n, a = 1, b = 10/7, n^{\log_b a} = n^0$$
$$\therefore \text{Complexity is } \Theta(n)$$

Validation:

$$\begin{aligned} T(n) = n &\le 7n/10 + n \\ &= 17/10n \end{aligned}$$

c. From master theorem:

$$f(n) = n, a = 4, b = 16, n^{\log_b a} = n^2$$
$$\therefore \text{Complexity is } \Theta(n^2 \lg n)$$

Validation:

$$\begin{aligned} T(n) = n^2 \lg n &\le 16c(n/4)^2 \lg(n/4) + n^2 \\ &= n^2 c \lg n - n^2, \forall c > 2 \text{ holds.} \end{aligned}$$

d. From master theorem:

$$f(n) = n^2, a = 7, b = 3, n^{\log_b a} < n^2$$
$$\therefore \text{Complexity is } \Theta(n^2)$$

Validation:

$$\begin{aligned} T(n) = n^2 &\le 7(n/3)^2 + n^2 \\ &= \frac{9}{16} n^2 \end{aligned}$$

e. From master theorem:

$$f(n) = n^2, a = 7, b = 2, n^{\log_b a} > n^2$$
$$\therefore \text{ Complexity is } \Theta(n^{lg7})$$

Validation:

$$\begin{aligned} T(n) = n^{lg7} &\leq 7(n/2)^{lg7} + n^2 \\ &= n^{lg7} + n^2 \end{aligned}$$

f. From master theorem:

$$f(n) = n^{\frac{1}{2}}, a = 2, b = 4, n^{\log_b a} = n^{\frac{1}{2}}$$
$$\therefore \text{ Complexity is } \Theta(\sqrt{n}\lg n)$$

Validation:

$$\begin{aligned} T(n) = \sqrt{n}\lg n &\leq 2\sqrt{(n/4)}\lg(n/4) + \sqrt{n} \\ &= \sqrt{n}c\lg n - \sqrt{n}, \forall c > 2 \text{ holds.} \end{aligned}$$

g.

The cost for each node: cn^2

The height of the tree: $n/2$

$$\begin{aligned} T(n) &= \sum_{i=1}^{n/2}(2i)^2 \\ &= 2^2 + 4^2 + 6^2 + \cdots + n^2 \\ &= 4\left(1^2 + 2^2 + 3^2 + \cdots + \left(\frac{n}{2}\right)^2\right) \\ &= 4 \cdot \frac{\left(\frac{n}{2}\right)\left(\frac{n}{2}+1\right)(n+1)}{6} \\ &= \Theta\left(n^3\right) \end{aligned}$$

8.1-3

From theorem 8.1, a decision tree with height h, l reachable nodes handling sorting problems with n! inputs, we have:

$$n! < l < 2^h$$

Therefore

$$h \geq lg(n!) = \Omega(n\lg n)$$

For half of the inputs:

$$h \geq lg(1/2n!) = \Omega(n\lg n) - 1$$

For 1/n of the inputs:

$$h \geq lg(n!/n) = \Omega(n \lg n) - lgn$$

For $1/2^n$ of the inputs:

$$h \geq lg(n!/2^n) = \Omega(n \lg n) - n$$

All of the above results give a complexity of $\Omega(n \lg n)$, so there doesn't exist comparison sort whose running tine is linear.

8.3-4

```
LINEAR_SORT(A):
    // convert the array into base n so that it has most 3 digits
    let B = CONVERT_TO_BASE_N(A)
    for i = 1 to 3
        // use counting sort to sort the ith digit of the array
        COUNTING_SORT(A, i)
```

8.4-3

Because when all element falls into one single bucket, bucket sort will then use insertion sort. Which gives a complexity of $O(n^2)$. To improve the worst case complexity into $O(n \lg n)$, we can us merge sort or quick sort to sort elements inside the same bucket.

9.3-3

If every time when choosing a pivot that is the median of the sub array with `SELECT` algorithm, the complexity of quick sort will be the worst case, which is $O(n \lg n)$.