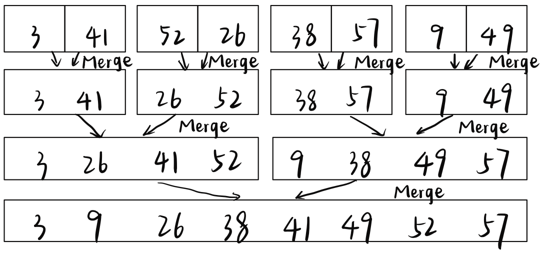
2.3-1

Using Figure 2.4 as a model, illustrate the operation of merge sort on the array $A = \langle 3, 41, 52, 26, 38, 57, 9, 49 \rangle$.

Solution:



2-4 *Inversions*

Let $A[1..n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an ***inversion*** of A.

a. List the five inversions of the array $\langle 2, 3, 8, 6, 1 \rangle$.

b. What array with elements from the set $\{1, 2, \ldots, n\}$ has the most inversions? How many does it have?

c. What is the relationship between the running time of insertion sort and the number of inversions in the input array? Justify your answer.

d. Give an algorithm that determines the number of inversions in any permutation on n elements in $\Theta(n \lg n)$ worst-case time. (*Hint:* Modify merge sort.)

a. the five inversions: (1, 5). (2, 5). (3, 5). (4. 5). (3. 4)

b. Among, the array $< n, n-1, n-2, \cdots, 2, 1 >$ has the most inversions and it has totally $(n-1)+(n-2)+\cdots+1 = \frac{n(n-1)}{2}$ inversions.

c. the running time would be longer when the number of inversions is larger. because in the loop of inverse sorting, the steps of moving when sorting the ith number of array, equals the number in $i \to (0, i-1)$ that greater than the ith number, which equals the numble of inversions pairs.

d. Using Merge Sorting method, pairing the adjacent elements groups by groups, and repeating;

then we have: $T(N) = \begin{cases} C & (n=1) \\ 2T(\frac{N}{2}) + CN & (n>1) \end{cases} \Rightarrow$ where $T(N) = \Theta(N \log N)$

the code of algorithm is as below

```
public static void Count_Inversions(int[]a){
  int[]tmpArray = new int[a.length];
  return count(a, tmpArray, 0, a.length-1);
}
private static void Count_Invertions(int[]a, int[]tmpArray, int left, int right){
  count = 0;
```

```
    if(left < right){
      int center = (left + right)/2;
      count = count + Count_Inversions(a, tmpArray, left, center);
      count = count + Count_Inversions(a, tmpArray, center+1, right);
      count = count + c_merge(a, tmpArray, left, center+1, right);
    }
    else {
      return 0;
    }
    return count;
  }
  private static void merge(int[]a, int[]tmpArray, int leftPos, int rightPos, int rightEnd){
    count = 0
    int leftEnd = rightPos - 1, tmpPos = leftPos;
    int numElements = rightEnd - leftPos + 1;
    while(leftPos <= leftEnd && rightPos <= rightEnd)
      if(a[leftPos] <= a[rightPos])
        temArray[tmpPos++] = a[leftPos++];
      else
        temArray[tmpPos++] = a[rightPos++]
        count++;
    while(leftPos <= leftEnd)
      tmpArray[tmpPos++] = a[leftPos++]
      count++;
    while(rightPos <= rightEnd)
      tmpArray[tmpPos++] = a[rightPos++];
    for(int i = 0; i < numElements; i++, rightEnd--)
      a[rightEnd] = empArray[rightEnd];
    return count
  }
```

3.1-1

Let $f(n)$ and $g(n)$ be asymptotically nonnegative functions. Using the basic definition of Θ-notation, prove that $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.

Solution: to prove $\max(f(n), g(n)) = \Theta(f(n)+g(n)) \Rightarrow \begin{cases} \max(f(n), g(n)) = O(f(n)+g(n)) \\ \max(f(n), g(n)) = \Omega(f(n)+g(n)) \end{cases}$

1° $O(f(n)+g(n))$

for $f(n)$ and $g(n)$ are non-negative functions.

$\max(f(n), g(n)) \leq (f(n)+g(n)) \times 1$ for any n.

$\max(f(n), g(n)) = O(f(n)+g(n))$

2° $\Omega(f(n)+g(n))$

$\max(f(n), g(n)) = \frac{1}{2}(f(n)+g(n) - |f(n)-g(n)|) \geqslant \frac{1}{2}(f(n)+g(n))$

$\max(f(n), g(n)) \geqslant (f(n)+g(n)) \times \frac{1}{2}$ for any n.

$\max(f(n), g(n)) = \Omega(f(n)+g(n))$

So, it can conclude that $\max(f(n), g(n)) = \Theta(f(n)+g(n))$

3.1-2

Show that for any real constants a and b, where $b > 0$,

$$(n + a)^b = \Theta(n^b) . \tag{3.2}$$

Solution:

1° prove $(n+a)^b = O(n^b)$:

when $n \geq 2|a|$: $(n+a)^b \leq (n+|a|)^b \leq (\frac{3}{2}n)^b$

$(n+a)^b \leq (\frac{3}{2})^b (n)^b$

$(n+a)^b = O(n^b)$

2° prove $(n+a)^b = \Omega(n^b)$:

when $n \geq 2|a|$: $(n+a)^b \geq (n-|a|)^b \geq (\frac{1}{2}n)^b$

$(n+a)^b \geq (\frac{1}{2})^b (n)^b$

$(n+a)^b = \Omega(n^b)$

so: it conclude that $(n+a)^b = \Theta(n^b)$ for $b>0$.

3.1-6

Prove that the running time of an algorithm is $\Theta(g(n))$ if and only if its worst-case running time is $O(g(n))$ and its best-case running time is $\Omega(g(n))$.

Solution: 1° if:

Let it's Worst-case running time to be T_w. best-case running time be T_b.

Knowing that $T_w(n) = O(g(n))$, $T_b(n) = \Omega(g(n))$.

we have $0 \leq c_1 g(n) \leq T_b(n) \leq T(n) \leq T_w(n) \leq c_2 g(n)$ for $n > \max(n_b, n_w)$

so: $T(n) = \Theta(g(n))$ by definition

2° only if:

$T(n) = \Theta(g(n)) \Rightarrow$ so that $T(n) = O(g(n))$, $T(n) = \Omega(g(n))$

$T_b(n) \geq c_1 g(n)$ for $n > n_b$, $T_w(n) \leq c_2 g(n)$ for $n > n_w$.

so: $T_w(n) = O(g(n))$. and $T_b(n) = \Omega(g(n))$

so proved.

4.3-6

Show that the solution to $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$ is $O(n \lg n)$.

Solution: to prove $T(n) = 2T(\lfloor \frac{n}{2} \rfloor + 17) + n$ is $O(n\lg n)$

so there exist an $d \geq 0$ such that $T(n-d) \leq c(n-d)\lg(n-d)$

$$\begin{aligned} T(n) = 2T(\lfloor \tfrac{n}{2} \rfloor + 17) + n &\leq 2c(\lfloor \tfrac{n}{2} \rfloor + 17)\lg(\lfloor \tfrac{n}{2} \rfloor + 17) + n \\ &\leq c(n+34)\lg(\tfrac{n+34}{2}) + n \\ &\leq cn\lg n \end{aligned}$$

$c(n+34)\lg(\frac{n}{2}+17) + n - cn\lg n \leq 0$

$cn\lg(\frac{n}{2}+17)+34c\lg(\frac{n}{2}+17)+n-cn\lg n \leq 0.$

for $c \geqslant 2$. $n>34$ the unequation feasible.

$T(n) \leq (4c) \cdot n\lg n$

$T(n)=O(n\lg n)$

So proved that $T(n)$ is $O(n\lg n)$.

4.5-1

Use the master method to give tight asymptotic bounds for the following recurrences.

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

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

c. $T(n) = 2T(n/4) + n$.

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

Solution: (a). when $a=2$. $b=4$. $\lambda=0$

$\log_b a = \log_4 2 = \frac{1}{2} \geqslant \lambda$

$\therefore T(n) = \Theta(n^{\frac{1}{2}})$

(b) when $a=2$. $b=4$. $\lambda=\frac{1}{2}$

$\log_b a = \log_4 2 = \frac{1}{2} = \lambda$

$\therefore T(n) = \Theta(n^{\frac{1}{2}}\log n)$

(c) when $a=2$. $b=4$. $\lambda=1$

$\log_b a = \log_4 2 = \frac{1}{2} < \lambda = 1$

$\therefore T(n) = \Theta(n)$

(c) when $a=2$, $b=4$. $\lambda=2$

$\log_b a = \log_4 2 = \frac{1}{2} < \lambda = 2$

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

4.5-2

Professor Caesar wishes to develop a matrix-multiplication algorithm that is asymptotically faster than Strassen's algorithm. His algorithm will use the divide-and-conquer method, dividing each matrix into pieces of size $n/4 \times n/4$, and the divide and combine steps together will take $\Theta(n^2)$ time. He needs to determine how many subproblems his algorithm has to create in order to beat Strassen's algorithm. If his algorithm creates a subproblems, then the recurrence for the running time $T(n)$ becomes $T(n) = aT(n/4) + \Theta(n^2)$. What is the largest integer value of a for which Professor Caesar's algorithm would be asymptotically faster than Strassen's algorithm?

Solution:

$T(n) = aT(\frac{n}{4}) + \Theta(n^2)$

$a=a$. $b=4$. $f(n)=\Theta(n^2)$

$n^{\log_b a} = n^{\log_4 a} < n^{\log_2 7}$

$\log_2 a < \log_2 49$

$a<49$

$\therefore$ the largest integer of a is 48