周子龙 1851201

Answer Sheet:

1. 2, log*n*, *n*2/3, 4*n*2, 3*n*, *n*！
2. (1) *f* ( *n* ) *= Θ* ( *g* ( *n* ) )

For , thus f(n) and g(n) only differ in constant coefficient.

(2) *f* ( *n* )*= O* ( *g* ( *n* ) )

For and , thus the former expression has lower rank class then the latter.

(3) *f* ( *n* ) *= Ω* ( *g* ( *n* ) )

As L’hospital’s law points out, , (here we substitute with , for there is only a constant factor different, hence this will not cause any different in the final answer), therefore has a higher rank class than .

(4) *f* ( *n* ) *= Ω* ( *g* ( *n* ) )

, and itself has a lower rank than n.

(5) *f* ( *n* ) *= Θ* ( *g* ( *n* ) )

are both constant.

(6) *f* ( *n* ) *= Ω* ( *g* ( *n* ) )

, and both of them are not constant.

(7) *f* ( *n* ) *= Ω* ( *g* ( *n* ) )

(8) *f* ( *n* )*= O* ( *g* ( *n* ) )

Both of the expressions have an exponential growth order, but f(n) have a smaller base then g(n).

1. 1) n+6

Let be the running time of the second machine, given , compare the two equations above, we can get Hence, using the same algorithm and in the same given time, the second machine can solve a problem set of .

2) 8n

3) any given magnitude.

For T’(n) = 1/8, thus for any given magnitude of problem set, the second machine can solve within the given time.

1. **Algorithm** ImprovedBinarySearch(A[0…n-1], x)

//Implements of nonrecursive improved binary search

//Input: An array A[0…n-1] sorted in ascending order and a search key x.

//Output: If x is not in the array, return the biggest index i and the smallest index j, which A[i] < x < A[j]. If x is in the array, the return value i, j will be of the same value. In extreme cases where there is no bigger/smaller key value than x, algorithm will return -1.

i🡨0, j🡨n-1

**while** i < j **do**

mid 🡨 (i+j)/2

**if** A[mid] < x **do**

i 🡨 mid+1

**else if** A[mid] > x **do**

j 🡨 mid-1

**else**

i 🡨 mid

j 🡨 mid

**if** A[i] < x **do**

**return** j, -1

**else if** A[i] == x **do**

**return** i, i

**else do**

**return** i-1, i

The basic operation for above algorithm is comparison. For a presorted array A, the key may occur at any position, thus the possibility of finding it is , where n is the length of a given array.

Hence, we can get following conclusion.

1. If the key happens to lie in the middle of the array,

Cbest(n) = 1

1. If the key happens to be the first or the last value of the array, or even does not reside the given array, in such scenario

using 2k substitute n, we may obtain:

1. For average case,

