**1.**

|  |  |  |  |
| --- | --- | --- | --- |
|  | *TA(n) =* | *TB(n) =* | *Faster* |
| Case 1 | *Θ(n2)* | *Θ()* | B  |
| Case 2 | *Ο(n3)* | *Ω()* | A  |
| Case 3 | *Θ(logn)* | *Ο(loglogn)* | B  |
| Case 4 | *Θ(log3n)* | *Θ()* | A |
| Case 5 | *Ο(n2)* | *Ο(n2.31)* | U |
| Case 6 | *Ω(n2)* | *Ο(n2.5)* | U |
| Case 7 | *Ω(n3)* | *Ο(n2.81)* | B |
| Case 8 | *Θ()* | *Θ()* | A |

**2.(a)**

**find-k**(*A*[*p..r*])

if *r*=*p*+1, RETURN *p*

*mid*=⎣(*p*+*r*)/2⎦

if *A*[*mid*]=0 and *A*[*mid*+1]=1, RETURN *mid*

if *A*[*mid*]=0, **find-k**(*A*[*mid..r*])

 else **find-k**(*A*[*p..mid*])

This is similar to binary search: with a constant number of comparisons, we reduce the problem size by half: T(*n*)=T(*n*/2)+*c* ⇒ T(*n*) = *O*(log*n*)

**(b)**

*i*←1

while *A*[*i*]=0

 *i*←2*i*

**find-k**(*A*[*i/*2+1*..i*])

The while loop will stop when it finds a 1. Since each time we double the value of *i*, the while loop performs log*k* iterations. The first 1 occurs somewhere between the positions *A*[*i/*2+1] and *A*[*i*]. To find it, we call **find-k**(*A*[*i/*2+1*..i*]), which has cost log(*k*/2)*=* *O*(log*k*)*.* Therefore, the total cost is *O*(log*k*).

**4.(a)** Suppose an MST contains a dangerous edge . We remove *e* from , which breaks into two trees and . We look at the cycle that has as its longest edge. This cycle must connect to via another path, which must have an edge connecting and , with . We add to . This gives us another spanning tree with weight less than the original MST. This is a contradiction.

**(b)** Suppose . First delete and all edges with weight greater than from the graph. Then run a BFS or DFS starting from . If the BFS/DFS reaches , answer “yes”, otherwise answer “no”.

**5.(a)** Sort all files in the decreasing order of their length. Proof of optimality: Consider any order and any two consecutive jobs , . If , then we can swap their order. This swap will increase the cost of by , but will decrease the cost of by , so will decrease the expected cost.

**(b)** Sort all files according to the ratio . Proof of optimality: Consider any order and any two consecutive jobs , . If , then we swap their order. This swap will increase the cost of by , but will decrease the cost of by . The net increase of the expected cost is thus

**6.(a)** *T*(1) = 1, 

Base: T(1)=1<21

Hypothesis: *T*(*m*) < 2*m* ∀ 1≤*m*<*n*

Step:



The recurrence and its solution is similar to the first (i.e., non dynamic programming) solution to the rod-cutting problem

**(b)**

DP(*n*)

R[1]=10

for *j*==2 to *n* // *j* is the problem size

R[*j*] = 5 // *n*>1

for *i* == 1 to *j*-1 do

 R[*j*] =R[*j*] +3· R[*i*];

return R[*n*];

The running time is (*n*-1)+(*n*-2)+….1= O(*n2*)

**7.** Let be the number of node . Define to be the maximum sum achievable for picking nodes from the subtree below (and including) . Then

If any of ’s children or grandchildren doesn’t exist, the corresponding term is 0.

The base case is if is a leaf.

We solve this recurrence following the post-order of the binary tree (from leaves to the root).