Yifan Yang 40038814

1.

a.F

b.F

c.T

d.T

e.T

f. F

g.F

2.

IH: Assume T(k)<=ck-1/3√(k) for all k<n  
IS:

T(n)=9T(n/9)+ √n

<=9(cn/9)-1/3\*9\*1/3\*√n+√n

<=cn-√n+√n

<=cn

<=cn-1/3√n +1/3√n

Then we need cn-1/3√n +1/3√n>= cn-1/3√(n)

1/3√n>=0

it is true for all n>=1

Boundary:

T(1)=2<=c-1/3 -> c>=7/3

Thus, with c=3, then T(n)<= cn-1/3√(n) for all n>=1

Which belongs to O(n)

3.

1/

assume a1 to an is arrayList A, an+1 to ak is array B

cause K=O(1), sort last k numbers need time O(1)

Use binarySearch to insert every numbers of array B into arrayList A //k\*logn=logn

return arrayList(n);

2/ assume a1 to an+k is in same array

SELECT a(n) will take O(n+k)=O(n+logn)=O(n)

3/ assume a1 to an+k is in same array

SELECT a(n) will take O(n+k)=O(n+n^7/8)=O(n)

4.

The median of A will be the (n+1)/2 th smallest number, // the number of items in A is odd for simplicity, even is the same logic

Use SELECT to find median //O(n)

Then use SELECT to k/2 elements smaller than median, push them into stack SMALL //kO(n)

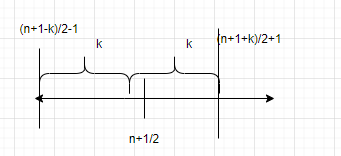
For example, median is 5th element, k=4. then find 4th element,3th element

use SELECT to k/2 elements bigger than median, push them into stack Big//kO(n)

if abs(small.top-median)<abs(big.top-median)

do{

compare abs(the next smallest number -median)and abs(big.top-median) // for example, the next smallest number is 2th element ,

if it is still smaller, push next smallest number into stack SMALL, Big.pop(); //O（1）

}

while abs(the next smallest number -median)> abs(big.top-median)

if abs(big.top-median)< abs(small.top-median)

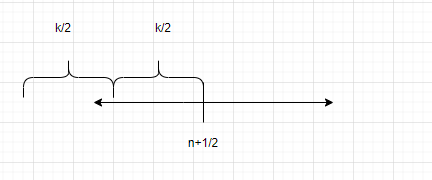
do{

compare abs(the next biggest number -median)and abs(small.top-median)

if it is still smaller, push next biggest number into stack BIG, Small.pop(); //O（1）

}

Even in most extreme situation, we only need to loop k/2 times, K\*O（1）



So total complexity is O(n)+kO(n)+kO(1)=O(n) //considering k is constant

5.

Assume the shortest path is V0 to Vn

and let current substructure vi to vj=path1

Assume there is path2 from vi to vj, shorter than path1

Length(path1 vi to vj)>Length(Path2 vi to vj)

Length(V0 toVi) + Length(path1 vi to vj)+Length(Vi to Vj)>

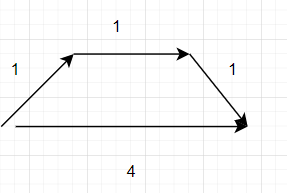
Length(V0 toVi) + Length(path2 vi to vj)+Length(Vi to Vj)

Current shortest path> Length(V0 toVi) + Length(path2)+Length(Vi to Vj)

Then it is not shortest path anymore, contradiction

b)

False



let k =2, now the path above will be 3+6=9

the path below will be 4+2=6,

now the path below is shorter

6.

PATH: ⌈n/3⌉

Cycle: ⌈n/3⌉

Complete Graph: 1

Complete bipartite graph:2

wheel graph :1

Complete binary tree:

7,(a).

Let G=(V,E)

C=empty\_set

E'=G.E

While E' !=empty

take a vertex v whose degree isn't 1 from bot to top

C=C+v

E=E'-the incident edges contains v //cost time most

}

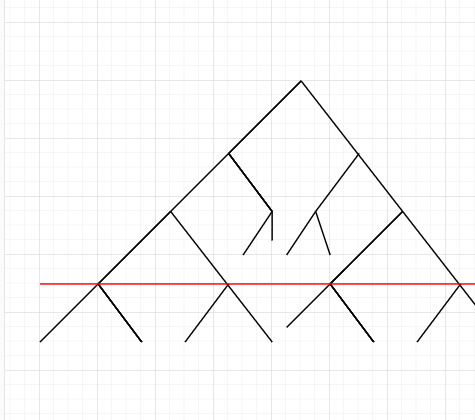
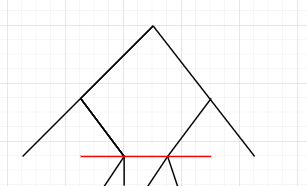
return C

cause every iteration will cost time proportional to the degree of V, so total time will be proportional to the sum of degrees=O（V）

(b)

Optimal substructure:

Cause we choose vertex v from bot to top

-> 

Its subproblem is still the same type of problem since we maintain the structure of tree

Greedy Property: At each time, we make best local decision, from depth (h-2) to depth (h),only if we remove the node in depth(h-1) whose degree is not 1, we can remove edges as much as possible

8.

a) Given a graph G=(V,E) and an integer k∈N, does there exist a set of non-adjacent vertices  V'⊆V of size |V'|≥k?

b)

STEP1: given the graph G, integer k and the set V', we can easily check is there a pair of adjacent nodes and size of V'>=k in polynomial time. So the problem is NP

STEP2: we need to reduce Clique problem to our problem of a) in polynomial time

Assume graph of clique problem is G(V,E)

we can construct graph G' in following way

V'=V all vertices of graph G(clique) is part of G'

E'=complement of edges E

Then G' is complementary graph of G

cause every vertices are connected in G, then every edges are not adjacent in G', for every edge in E', we pick a node of it.

Then