#1 Min and Max

def mm(a,l,h):

if l==h:

return (a[l],a[l])

elif h==l+1:

if a[l]>a[h]:

return (a[h],a[l])

return (a[l],a[h])

else:

mid=(l+h)//2

l1,m1=mm(a,l,mid)

l2,m2=mm(a,mid+1,h)

mi=min(l1,l2)

mx=max(m1,m2)

return (mi,mx)

a=[5,7,3,4,9,12,6,2]

l,h=0,len(a)-1

print(mm(a,l,h))

#2 Min and Max (Ascending array)

def mm(a,l,h):

if l==h:

return (a[l],a[l])

elif h==l+1:

if a[l]>a[h]:

return (a[h],a[l])

return (a[l],a[h])

else:

mid=(l+h)//2

l1,m1=mm(a,l,mid)

l2,m2=mm(a,mid+1,h)

mi=min(l1,l2)

mx=max(m1,m2)

return (mi,mx)

a=[2,4,6,8,10,12,14,18]

l,h=0,len(a)-1

print(mm(a,l,h))

#3 Merget Sort

def m(a):

if len(a)<=1:

return a

mid=len(a)//2

l=m(a[:mid])

r=m(a[mid:])

i,j=0,0

b=[]

while i<len(l) and j<len(r):

if l[i]<r[j]:

b.append(l[i])

i+=1

else:

b.append(r[j])

j+=1

while i<len(l):

b.append(l[i])

i+=1

while j<len(r):

b.append(r[j])

j+=1

return b

a=[31,23,35,27,11,21,15,28]

print(m(a))

#4 Merge Sort

def m(a):

if len(a)<=1:

return a

mid=len(a)//2

l=m(a[:mid])

r=m(a[mid:])

i,j=0,0

b=[]

while i<len(l) and j<len(r):

if l[i]<r[j]:

b.append(l[i])

i+=1

else:

b.append(r[j])

j+=1

while i<len(l):

b.append(l[i])

i+=1

while j<len(r):

b.append(r[j])

j+=1

return b

a=[12,4,78,23,45,67,89,1]

print(m(a))

#5 Quick Sort

def swap(a, i, j):

temp = a[i]

a[i] = a[j]

a[j] = temp

def par(a, low, high):

p = a[high]

i = low - 1

for j in range(low, high):

if a[j] < p:

i += 1

swap(a, i, j)

swap(a, i + 1, high)

return i + 1

def quick(a, low, high):

if low < high:

x = par(a, low, high)

quick(a, low, x - 1)

quick(a, x + 1, high)

a = [3, 5, 2, 6, 7, 9, 1]

n=len(a)

low,high = 0,n - 1

quick(a, low, high)

print(a)

#6 Quick Sort

def swap(a, i, j):

temp = a[i]

a[i] = a[j]

a[j] = temp

def par(a, low, high):

p = a[high]

i = low - 1

for j in range(low, high):

if a[j] < p:

i += 1

swap(a, i, j)

swap(a, i + 1, high)

return i + 1

def quick(a, low, high):

if low < high:

x = par(a, low, high)

quick(a, low, x - 1)

quick(a, x + 1, high)

a = [19,72,35,46,58,91,22,31]

n=len(a)

low,high = 0,n - 1

quick(a, low, high)

print(a)

#7 Binary Search

def b(a,t):

n=len(a)

l,h=0,n-1

while l<=h:

mid=(l+h)//2

if a[mid]==t:

return mid

elif a[mid]<t:

l=mid+1

else:

h=mid-1

return None

a=[5,10,15,20,25,30,35,40,45]

t=20

print(b(a,t))

#8 Binary Search for non sorted array

def unb(a, t):

l, h = 0, len(a) - 1

while l <= h:

m = (l + h) // 2

if a[m] == t:

return m

if a[l] <= a[m]:

if a[l] <= t < a[m]:

h = m - 1

else:

l = m + 1

else:

if a[m] < t <= a[h]:

l = m + 1

else:

h = m - 1

return -1

a = [4,5,6,7,0,1,2]

t = 0

print(unb(a,t))

#9 Kth minimum closest point to origin

from math import sqrt

def cl(a,n,k):

b=[]

min=float('inf')

for i in range(n):

x=sqrt((a[i][1]-0)\*\*2 + (a[i][0]-0)\*\*2)

b.append([x,a[i]])

b.sort()

return b[k-1]

a= [[1,3],[-2,2]]

k=1

n=len(a)

print(cl(a,n,k))

#10 Four Tuple sum to zero

def f(A, B, C, D):

m = {}

c = 0

for i in range(len(A)):

for j in range(len(B)):

s = A[i] + B[j]

if s in m:

m[s] += 1

else:

m[s] = 1

for k in range(len(C)):

for l in range(len(D)):

t = -(C[k] + D[l])

if t in m:

c += m[t]

return c

a1 = [1, 2]

b1 = [-2, -1]

c1 = [-1, 2]

d1 = [0, 2]

print(f(a1, b1, c1, d1))

#11 Median of median

def m(a, k):

def p(a, l, r):

x = a[r]

i = l - 1

for j in range(l, r):

if a[j] <= x:

i += 1

a[i], a[j] = a[j], a[i]

a[i + 1], a[r] = a[r], a[i + 1]

return i + 1

def h(a, l, r, k):

if l <= r:

if l == r:

return a[l]

n = r - l + 1

g = (n + 4) // 5

m = []

for i in range(g):

gl = l + i \* 5

gr = min(gl + 4, r)

mi = p(a, gl, gr)

m.append(a[mi])

pivot = h(m, 0, len(m) - 1, len(m) // 2)

pivot\_index = a.index(pivot)

pivot\_index = p(a, l, r)

if k == pivot\_index:

return a[pivot\_index]

elif k < pivot\_index:

return h(a, l, pivot\_index - 1, k)

else:

return h(a, pivot\_index + 1, r, k)

n = len(a)

if k >= 1 and k <= n:

return h(a, 0, n - 1, k - 1)

else:

raise ValueError("k is out of range")

arr = [12, 3, 5, 7, 19]

k = 2

print(m(arr, k))

#12

def m(a, k):

def p(a, l, r):

x = a[r]

i = l - 1

for j in range(l, r):

if a[j] <= x:

i += 1

a[i], a[j] = a[j], a[i]

a[i + 1], a[r] = a[r], a[i + 1]

return i + 1

def h(a, l, r, k):

if l <= r:

if l == r:

return a[l]

n = r - l + 1

g = (n + 4) // 5

m = []

for i in range(g):

gl = l + i \* 5

gr = min(gl + 4, r)

mi = p(a, gl, gr)

m.append(a[mi])

pivot = h(m, 0, len(m) - 1, len(m) // 2)

pivot\_index = a.index(pivot)

pivot\_index = p(a, l, r)

if k == pivot\_index:

return a[pivot\_index]

elif k < pivot\_index:

return h(a, l, pivot\_index - 1, k)

else:

return h(a, pivot\_index + 1, r, k)

n = len(a)

if k >= 1 and k <= n:

return h(a, 0, n - 1, k - 1)

else:

raise ValueError("k is out of range")

arr = [1,2,3,4,5,6,7,8,9,10]

k = 6

print(m(arr, k))

#13 Meet in middle

def get\_subsets(arr):

subsets = []

n = len(arr)

for i in range(1 << n):

subset\_sum = 0

for j in range(n):

if i & (1 << j):

subset\_sum += arr[j]

subsets.append(subset\_sum)

return subsets

def meet\_in\_the\_middle(arr, key):

n = len(arr)

left\_half = arr[:n//2]

right\_half = arr[n//2:]

left\_sums = get\_subsets(left\_half)

right\_sums = get\_subsets(right\_half)

right\_sums\_set = set(right\_sums)

for sum\_left in left\_sums:

if key - sum\_left in right\_sums\_set:

return True

return False

a = [1, 3, 9, 2, 7, 12]

key = 15

print(meet\_in\_the\_middle(a, key))

#14

def get\_subsets(arr):

subsets = []

n = len(arr)

for i in range(1 << n):

subset\_sum = 0

for j in range(n):

if i & (1 << j):

subset\_sum += arr[j]

subsets.append(subset\_sum)

return subsets

def meet\_in\_the\_middle(arr, key):

n = len(arr)

left\_half = arr[:n//2]

right\_half = arr[n//2:]

left\_sums = get\_subsets(left\_half)

right\_sums = get\_subsets(right\_half)

right\_sums\_set = set(right\_sums)

for sum\_left in left\_sums:

if key - sum\_left in right\_sums\_set:

return True

return False

a = [1, 3, 9, 2, 7, 12]

key = 15

print(meet\_in\_the\_middle(a, key))

#15 Strassen's Multiplication

def amat(a, b):

n = len(a)

c = [[0 for \_ in range(n)] for \_ in range(n)]

for i in range(n):

for j in range(n):

c[i][j] = a[i][j] + b[i][j]

return c

def smat(a, b):

n = len(a)

c = [[0 for \_ in range(n)] for \_ in range(n)]

for i in range(n):

for j in range(n):

c[i][j] = a[i][j] - b[i][j]

return c

def multiply(a, b):

n = len(a)

if n == 1:

return [[a[0][0] \* b[0][0]]]

mid = n // 2

a11 = [[a[i][j] for j in range(mid)] for i in range(mid)]

a12 = [[a[i][j] for j in range(mid, n)] for i in range(mid)]

a21 = [[a[i][j] for j in range(mid)] for i in range(mid, n)]

a22 = [[a[i][j] for j in range(mid, n)] for i in range(mid, n)]

b11 = [[b[i][j] for j in range(mid)] for i in range(mid)]

b12 = [[b[i][j] for j in range(mid, n)] for i in range(mid)]

b21 = [[b[i][j] for j in range(mid)] for i in range(mid, n)]

b22 = [[b[i][j] for j in range(mid, n)] for i in range(mid, n)]

m1 = multiply(amat(a11, a22), amat(b11, b22))

m2 = multiply(amat(a21, a22), b11)

m3 = multiply(a11, smat(b12, b22))

m4 = multiply(a22, smat(b21, b11))

m5 = multiply(amat(a11, a12), b22)

m6 = multiply(smat(a21, a11), amat(b11, b12))

m7 = multiply(smat(a12, a22), amat(b21, b22))

c11 = amat(smat(amat(m1, m4), m5), m7)

c12 = amat(m3, m5)

c21 = amat(m2, m4)

c22 = amat(smat(amat(m1, m3), m2), m6)

c = [[0 for \_ in range(n)] for \_ in range(n)]

for i in range(mid):

for j in range(mid):

c[i][j] = c11[i][j]

c[i][j + mid] = c12[i][j]

c[i + mid][j] = c21[i][j]

c[i + mid][j + mid] = c22[i][j]

return c

a = [[1,7],[3,5]]

b = [[6,8],[4,2]]

for k in a:

print(k)

print()

for k in b:

print(k)

c = multiply(a, b)

print()

for k in c:

print(k)

#16 Karatsuba Multiplication

def k(x, y):

def d(n):

c = 0

while n != 0:

n //= 10

c += 1

return c

def s(n, m):

h = n // (10 \*\* m)

l = n % (10 \*\* m)

return h, l

if x < 10 or y < 10:

return x \* y

n = max(d(x), d(y))

m = n // 2

h1, l1 = s(x, m)

h2, l2 = s(y, m)

z0 = k(l1, l2)

z1 = k(l1 + h1, l2 + h2)

z2 = k(h1, h2)

return (z2 \* 10 \*\* (2 \* m)) + ((z1 - z2 - z0) \* 10 \*\* m) + z0

a = 1234

b = 5678

print(k(a,b))