**1.**

import random  
  
def pivot\_firstitem(arr, low, high):  
 global first  
 if len(arr) <= 1:  
 return arr  
 pivotitem = arr[0]  
 j = low  
 for x in range(low+1, high+1):  
 first += 1  
 if(arr[x] < pivotitem):  
 j += 1  
 arr[x], arr[j] = arr[j], arr[x]  
 arr[low], arr[j] = arr[j], arr[low]  
 return pivot\_firstitem(arr[0:j], 0, j-1) + [arr[j]] + pivot\_firstitem(arr[j+1:], 0, len(arr[j+1:])-1)  
  
def pivot\_randomitem(arr, low, high):  
 global randompick  
 if len(arr) <= 1:  
 return arr  
 idx = random.randrange(low, high+1)  
 pivotitem = arr[idx]  
 if idx != 0:  
 arr[0], arr[idx] = arr[idx], arr[0]  
 j = low  
 for x in range(low+1, high+1):  
 randompick += 1  
 if(arr[x] < pivotitem):  
 j += 1  
 arr[x], arr[j] = arr[j], arr[x]  
 arr[low], arr[j] = arr[j], arr[low]  
 return pivot\_randomitem(arr[0:j], 0, j-1) + [arr[j]] + pivot\_randomitem(arr[j+1:], 0, len(arr[j+1:])-1)  
  
def pivot\_Robert(arr, low, high):  
 global robert  
 if len(arr) <= 1:  
 return arr  
 mid = (low+high)//2  
 pivotitem = arr[low]  
 mx = max(arr[low], arr[high], arr[mid])  
 mn = min(arr[low], arr[high], arr[mid])  
 if arr[low] != mx and arr[low] != mn:  
 pivotitem = arr[low]  
 elif arr[mid] != mx and arr[mid] != mn:  
 pivotitem = arr[mid]  
 elif arr[high] != mx and arr[high] != mn:  
 pivotitem = arr[high]  
 elif arr[mid] == arr[high] and arr[high] > arr[low]:  
 pivotitem = arr[high]  
 j = low  
 if arr.index(pivotitem) != 0:  
 idx = arr.index(pivotitem)  
 arr[low], arr[idx] = arr[idx], arr[low]  
 for x in range(low + 1, high + 1):  
 robert += 1  
 if (arr[x] < pivotitem):  
 j += 1  
 arr[x], arr[j] = arr[j], arr[x]  
 arr[low], arr[j] = arr[j], arr[low]  
 return pivot\_Robert(arr[0:j], 0, j - 1) + [arr[j]] + pivot\_Robert(arr[j+1:], 0, len(arr[j+1:]) - 1)  
  
  
first = 0  
randompick = 0  
robert = 0for y in range(5):  
 first = 0  
 randompick = 0  
 robert = 0  
 inp = [random.randint(1, 1000000) for x in range(500000)]  
 pivot\_firstitem(inp, 0, len(inp)-1)  
 pivot\_randomitem(inp, 0, len(inp)-1)  
 pivot\_Robert(inp, 0, len(inp)-1)  
 print(first, randompick, robert)

**2.**

I tried 5 times each array size.





**3.**

case1 depends on first element value. If first element is closer to the median, its time complexity is decreased. But If it is ordered array, time complexity(compare time) is increased. Because partition is unbalanced and array is biased.

case2 depends on what I choose. There’s a big difference in compare number depending on which number is selected. If pivot is closer to the median, it maybe best time complexity because its partition is balanced.

case3 is the best pivot of three I think. Because It compares 3 elements and chooses median. So probability of being close to the median in the entire array is increased. Sometimes If 3 elements(first, last, median) of the array is closer to the big or small, its compare time would be increased. But, I think ‘that probability’ is lesser than the probability of being close to the median. And in my experiment, most of the compare times of case 3 is lesser than case 1 and case2.

Based on my experiment, I prefer case3 for selecting pivot. Because most of the case3’s compare times is less than case 1 and case 2. Also case2 is unstabled in time complexity becase of ‘random’ pivot. And case 1 is more stabled than case 2. But, if array is ordered list or pivot is close to big or small element, partition is unbalanced. So I select case3. Sometimes, case 1 is faster than case3. But, In terms of stability, I think case 3 is most stabled. Because it compare three elements to select pivot.

I thought, In case 3, if I increase the number of pivot to compare, the probability of being close to median is increased. But it takes more time than case 3 in terms of choosing median number if array size is not large enough.