CSE 101: Design and Analysis of Algorithms **WI 2016**

Programming Assignment #1 Solutions

Qian YingDong A92501011

Problem 5

Algorithm.

Input: a rotated sorted array and the desired element

Output: the index of desired element

We use divide and conquer method for this problem. By considering the gratitude among the first element(a), midpoint element(c) and the desired element(b), we break the original problem into 6 sub problems.

1. $a > b > c$, we apply binary search to the second half of the array.
2. $a > c > b$, we apply rotated sorted array search method to the first half of the array.
3. $b > a > c$, we apply rotated sorted array search method to the first half of the array.
4. $b > c > a$, we apply rotated sorted array search method to the second half of the array.
5. $c > a > b$, we apply rotated sorted array search method to the second half of the array.
6. $c > b > a$, we apply binary search to the first half of the array.

Through these 6 subproblems we can solve the problem.

Complexity

Using either the rotated sorted array search or the binary search, we can halve the problem size within constant numbers of comparisons. So the time complexity is $O(\log n)$.

Analysis

From table 1 in the next page, we can easily see the great advantage of DQ over naive method. Since naive method is $O(n)$ and DQ method is $O(\log n)$, we can find the growing speed of time cost when we increase the array size by 100 each time. So we can have a basic conclusion that time complexity is a key factor as the input size grows very large.

notes:

Due to the limitation of int type, the largest array size tested here is 10^9.

Since the time for DQ is always $0ms$*, I change the unit to* μs

Table 1: Naive v.s. DQ

array sizes	**naive**(μs)	**DQ**(μs)
10^3	5	1
	3	1
	3	1
	3	1
	2	1
average:	3.2	1
10^5	262	1
	258	1
	273	2
	379	2
	410	2
average:	316.4	1.6
10^7	6430	3
	13079	3
	13357	3
	22893	4
	33670	4
average:	17885.8	3.4
10^9	4660905	7
	2215687	8
	3729655	7
	581853	7
	2256242	11
average:	2688868.4	8