|  |  |
| --- | --- |
|  | Quick Sort Information Handout |
| Visual Exemplar | Method |
| http://www.java2novice.com/images/quick-sort1.png | * Select pivot value * Compare all elements of the array until all elements greater than the pivot are on the left and all elements smaller than the pivot are on the right * Place the pivot on the wall, partitioning the array into two subarrays * Apply quick sort recursively to the sub arrays until all elements are sorted |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NEW TERMINOLOGY** | **Pivot**: An element of the array that is compared with all the other elements  **Wall**: A divider used to partition an array or subarray in the quick sorting method  **Current Element**: The element that is being compared to the pivot  **Partition**: Dividing an array into different parts  **Subarray**: A subsection of an array | | | |
| **EFFICIENCY** | Picking the Pivot:   * Last/First Element (highest chance of worst case) * Randomly (lower chance of worst case) * Median of the first, last, & middle elements (minimizes worst case)   Big O   * Worst Case: O(n2) * Best Case: O(nlogn) | | | |
| **WHY** | | **WHEN** | | **WHERE** |
| * Cache efficient & usually fast * High chance to pick a number that can partition the array into 2 parts | | * No need for stable sort * Average performance is more important | | * Used to sort arrays * Used for randomized arrays |
| **PROS** | | | **CONS** | | |
| * Recursive method is easy to implement * On paper, more efficient than all other sorting methods * More memory efficient than merge sort * Best case sorting is O(nlogn) * Extremely efficient if right pivot is chosen * Able to deal with large array sizes | | | * In the worst case, can be as slow as bubble sort (n2) * Iterative implementation is difficult to implement (faster than recursive) * Is an unstable sorting method (Does not keep things in relative order) | | |