# Sort Algorithms Review

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Description | Stable | In Place | Extra Space | Worst case | Average | Best | Keys  Type | **Comparison**  **Sorts** |
| Insert every element to its correct place in the left sub array by  shifting elements to th right to make room for the inserted value.  At any given time the sub array to the left of the value being  inserted is always sorted and to the right usually not sorted.  Is fast for n< 50 and also on a nearly sorted array. | √ | √ | O(1) | Θ (n2) | Θ (n2) | Θ(n) | any | Insertion Sort  הכנסה |
| In each pass compare pairs and swap the values  if needed. | √ | √ | O(1) | Θ (n2) | Θ (n2) | Θ(n) | any | Bubble Sort  בועות |
| In each pass swaps the minimal value with the first  value in the sub array being dealt with. | X | √ | O(1) | Θ (n2) | Θ (n2) | Θ (n2) | any | Selection Sort  בחירה |
| Divide the array into 2 equal parts. Recusuvely sort  each part and then merge the two parts. | √ | X | Θ (n) | Θ (nlogn) | Θ (nlogn) | Θ (nlogn) | any | Merge Sort  מיזוג |
| Build a Heap from the values. Then for each value  in the heap place the maximum at the end of the  array then replace the maximum with the last leaf  and perform a heapify to rectify the heap. | X | √ | O(1) | O(nlogn) | O(nlogn) | O(nlogn) | any | Heap Sort  ערמה |
| Partition the array according to a pivot value, not  necessarily to two equal parts and the recursively  sort each part. | X | √ | O(1) | Θ (n2) | Θ (nlogn) | Θ (nlogn) | any | Quick Sort  מהיר |
| Builds an AVL tree and the does an inorder transversal | √  Assuming equal values are placed to the right | X | Θ (n) | Θ (nlogn) | Θ (nlogn) | Θ (nlogn) | any | Binary Sort  AVL  בינארי  Tree Sort |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Description | Stable | In Place | Extra Space | All Cases | Keys  Type | **Counting**  **Sorts** |
| For every element x in the array count how elements are less than or equal to it. Places x directly to its correct place.  **If k is less than n then the complexity is** Θ (n) | √ | X | Θ(n+k) | Θ(n+k) | integers [0..k] | Counting Sort  מניה |
| Works on whole numbers in base b with d digits.  Sort the number d times, each time according to a different digit starting from the LSD. | √ | Depends on the stable sort used | Depends on the stable sort used | Θ (d(b+n))  Θ (n) | d digits in base b | Radix Sort  בסיס |
| Divides the range into n sub-ranges or buckets. Each element is put in its appropriate bucket and then each bucket is sorted. After that concatenate the buckets. | √ | X | Θ (n) | Θ (n)  If not uniform  Θ (n2) | [0,1)  [0,k)  Assumes uniform  distrubution | Bucket Sort  דלי  Bin Sort |

|  |  |  |
| --- | --- | --- |
| **Counting-Sort**(A)  for i=0 to k C[i]=0  for j=1 to n C[A[j]]=C[A[j]]+1  for i=1 to k C[i]=C[i]+C[i-1]  for **j=n downto 1**  B[C[A[j]]]=A[j]  C[A[j]]=C[A[j]]-1 | Radix-Sort(A,b,d) // LSDfor i=1 to duse a stable sort (like counting sort)to sort Aaccording to the i’th digit in base b | Bucket-Sort(A)n=length[A]for i=1 to ninsert A[i] into bucket B[nA[i]]for i=0 to n-1sort bucket B[i] using Insertion-SortC= concatenate buckets B[0],B[1],…,B[n-1] |