**Lab 7**

**Solam Jung Rana (986888)**

1. Show that any comparison-based algorithm to sort 4 elements requires at least 5 comparisons.

**Solution:**

|  |  |  |  |
| --- | --- | --- | --- |
| **2, 0 || 3, 1 → 0, 1, 2, 3** | | | |
|  | | | |
| **2 || 0 → 0, 2** | | **3 || 1 → 1, 3** | |
|  | | | |
| **2** | **0** | **3** | **1** |

**Comparison 1: 0 and 2**

**Comparison 2: 3 and 1**

**Comparison 3 between (0, 2) and (1, 3): 0 and 1**

**Comparison 4 between (2) and (1, 3): 2 and 1**

**Comparison 5 between (2) and (3): 2 and 3**

1. Devise an algorithm to sort 4 elements using exactly 5 comparisons.

**Solution:**

**Algorithm Sort(S)**

**Input: Sequence S with n values**

**Output: Sequence S sorted**

**if S.size() > 1 then**

**(S1, S2) ← partition(S, n/2)**

**Sort(S1)**

**Sort(S2)**

**S ← merge(S1, S2)**

**return S**

1. Devise an algorithm that arranges the elements of a length**-**n integer array according to the following scheme:

position 0: the smallest integer

position 1: the largest integer

position 2: the second smallest integer

position 3: the second largest integer etc.

For example, this algorithm would arrange the input array {1, 2, 17, -4, **-**6, 8} as follows: {-6, 17, -4, 8, 1, 2}. (Notice that –6 is the smallest, 17 the largest, -4 second smallest, 8 second largest, etc.) What is the asymptotic running time of your algorithm? What is the fastest possible asymptotic running time for such an algorithm? Prove your answer.

**Solution:**

**Algorithm SpecialSort(A)**

**Input: A is an integer array**

**Output: A as a specially sorted array**

**A ← MergeSort(A)**

**B ← new Array(A.size())**

**mid ← A.size() / 2**

**(i, j) ← (0, A.size())**

**while i < mid then**

**B[i] ← A[i++]**

**while j ≥ mid then**

**B[j] ← A[j--]**

**A ← B**

**return A**

**Asymptotic running time: O(nlog n)**

**Fastest possible: O(n) when there exists an optimization and already sorted array.**

**Informal Proof:**

**Only one pass through the array with size of n that has already been sorted is needed.**

1. Use RadixSort to sort the following: {80, 1, 46, 53, 28, 55, 32, 6, 9}, with radix = 9.

**Solution:**

|  |
| --- |
| **In base 10:**  **{80, 1, 46, 53, 28, 55, 32, 6, 9}** |
| **To base 9:**  **{88, 1, 51, 58, 31, 61, 35, 6, 10}** |
| **First pass:**  **61**  **31**  **51 58**  **10 1 35 6 88** |
| **0 1 2 3 4 5 6 7 8** |
| **Second pass:**  **6 35 58**  **1 10 31 51 61 88** |
| **0 1 2 3 4 5 6 7 8** |
| **In base 9:**  **{1, 6, 10, 31, 35, 51, 58, 61, 88}** |
| **Back to base 10:**  **{1, 6, 9, 28, 32, 46, 53, 55, 80}** |