Selection between lists

Shashwat Gupta (14IE10028)

April 6, 2016

1 Finding Kth recursively

Finds the k/2 th smallest element in both arrays and then finds the smaller one. It then looks at elements to the right of the array with smaller k/2 th and left of the other. This is done recursively till the kth smallest is found.

```
int find_k_func(int arr1[], int arr2[], int n1, int n2, int k)
{
    int k2 = max(1, (k - 1) / 2);
    int a = sendKthSmallest(arr1, 0, n1 - 1, min(k2,n1));
    int b = sendKthSmallest(arr2, 0, n2 - 1, min(k2,n2));
    if (k < 2)
        return min(a, b);
    int last1 = min(n1, n1 );
    if (arr1[last1] == -10000 || arr1[last1] >(1 << 15));
    int last2 = min(n2, n2 );
    if (arr2[last2] == -10000 || arr2[last2] > (1 << 15));
    if (b == a)
        return find_k_func(arr1 + min(k2, n1), arr2 + min(k2, n2),
    n1 - min(k2, n1), n2 - min(k2, n2), k - k2 * 2);
    else if (b < a)
        return find_k_func(arr1, arr2 + min(k2, n2), last1, n2 -
    min(k2, n2), k - k2);
    else
        return find_k_func(arr1+min(k2, n1), arr2 , n1-min(k2, n1),
     last2, k - k2);
}
```

Listing 1: K recursive

2 Kth Smallest

The following is the code which calculates the kth smallest element in an array and returns it, without sorting the array. In lenier time.

```
int sendKthSmallest(int arr[], int l, int r, int k)
{
    if (k > 0 && k <= r - l + 1)
    {
        int n = r-l+1;
        int i, median[(n+4)/5];
```

```
        for (i=0; i<n/5; i++)
            median[i] = giveMedian(arr+l+i*5, 5);
        if (i*5 < n)
        {
            median[i] = giveMedian(arr+l+i*5, n%5);
            i++;
        }
        int mainMed = (i == 1)? median[i-1]:
        sendKthSmallest(median, 0, i-1, i/2);
        int pos = partition(arr, l, r, mainMed);
        if (pos-l == k-1)
            return arr[pos];
        if (pos-l > k-1)
            return sendKthSmallest(arr, l, pos-1, k);
        return sendKthSmallest(arr, pos+1, r, k-pos+l-1);
    }
    return 10000;
}
```

Listing 2: Kth smallest

3 Partitioning

The following is the code which partitions an array according to a given pivot. The array separates elements smaller than the pivot to its left and the elements larger to its right.

```
int partition(int arr[], int l, int r, int x)
{
    int i,j;
    for (i=l; i<r; i++)
        if (arr[i] == x)
           break;
    swap(&arr[i], &arr[r]);
    i = l;
    for (j = l; j < r; j++)
    {
        if (arr[j] <= x)
        {
            swap(&arr[i], &arr[j]);
            i++;
        }
    }
    swap(&arr[i], &arr[r]);
    return i;
}
```

Listing 3: Partitioning

4 Complexity

The time complexity of the code is $\in O(nlogk)$.