Master Mathmods, april 30, 2012

Searching and hashing

1 Selection

We consider the following procedure which akin to quicksort for finding the k-th element of an array if it were sorted but without sorting it. (The source is avaible for download from the website).

```
def kselect(k)
 if self.size >= k
  pivot=self[rand(self.size)]
  a=select{ |x| x < pivot}
  b=select{ |x| x == pivot}
  c=select{ |x| x> pivot }
  if a.size >= k then return a.kselect(k)
  elsif (a.size+b.size)>=k then return pivot
  else return c.kselect(k-a.size-b.size)
  end
 end
end
```

(1) Explain the behaviour of this algorithm

(2) What's its time complexity in the worst case?

(3) Show that it has a linear average execution time.

2 Hashing

(1) We are given a new sorting algorithm and we'd like to check its behavior for big data sets. We'd like to check:

- that the array is sorted;
- that the data haven't been corrupted.

Propose a method to fulfill the above requirements in a time linear in the array's size.

(2) Assume $h_1, h_2, \ldots$ is an infinite sequence of hash functions with values in $[\![1, m]\!]$ which are uniform and independant; we'd like to use them one after the other to avoid the clustering problem.

Let p_r be the probability that exactly r comparisons are required for finding an empty cell in a hash table of size m containing n elements.

(a) Prove that $p_r = \alpha^r(1-\alpha)$ where $\alpha = n/m$ denotes the hash-table filling rate;

(b) deduce that $\alpha/(1-\alpha)$ is the expected number of comparisons for a negative search;

(c) show that the expected number of comparisons for a positive search is $\left(\frac{1}{\alpha}\sum_{j=0}^{n-1}\frac{1}{m-j}\right) - 1 = \frac{m}{n}(H_m - H_n) - 1$ where H_i denotes the i-th harmonic number, sum of the reciprocal of the first i natural numbers $\sum_{k=1}^{i}\frac{1}{k}$.