SortTimes

kaymas

August 3, 2018

Complexity for different Sorting Algorithms.

Insertion Sort

Sorting Algorithm

```
insertionSort <- function(vec){
  n <- length(vec)
  for(i in 2:n){
    val <- vec[i]
    pos <- which.max(vec[1:i] > val) #returns index of first occurence of TRUE
    if(pos == 1){
      if(val < vec[1]){
        vec <- c(val, vec[-i])
      }
    }
    else{
      vec <- vec[-i]
      vec <- c(vec[1:(pos-1)], val, vec[pos:(n-1)])
    }
  }
  return (vec)
}
```

Proof of concept

```
insertionSort(c(1,2,99,-21,2,23,1))
```

```
## [1] -21   1   1   2   2  23  99
```

RunTime

```
system.time(replicate(10, insertionSort(sample(x = 1:100, size = 10, replace = TRUE)))) / 10
```

```
##    user  system elapsed
##   3e-04   0e+00   2e-04
```

```
system.time(replicate(10, insertionSort(sample(x = 1:100, size = 100, replace = TRUE)))) / 10
```

```
##    user  system elapsed
##   6e-04   0e+00   7e-04
```

```
system.time(replicate(10, insertionSort(sample(x = 1:100, size = 1000, replace = TRUE)))) / 10
```

```
##    user  system elapsed
##  0.0321  0.0000  0.0320
```

```
system.time(replicate(10, insertionSort(sample(x = 1:100, size = 10000, replace = TRUE)))) / 10
```

```
##    user  system elapsed
##  2.9139  0.0001  2.9157
```

```
#mean(replicate(10, system.time((insertionSort(sample(x = 1:100, size = 10000, replace = TRUE))))[3]))
```

Merge Sort

Sorting Algorithm

```
mergeSort <- function(vec){

  mergeTwo <- function(left,right){
    res <- c()
    while(length(left) > 0 && length(right) > 0){
      if(left[1] <= right[1]){
        res <- c(res,left[1])
        left <- left[-1]
      }else{
        res <- c(res,right[1])
        right <- right[-1]
      }
    }
    if(length(left) > 0) res <- c(res,left)
    if(length(right) > 0) res <- c(res,right)
    return (res)
  }

  n <- length(vec)
  if(n <= 1) return (vec)
  else{
    middle <- length(vec) / 2
    left <- vec[1:floor(middle)]
    right <- vec[floor(middle + 1):n]
    left <- mergeSort(left)
    right <- mergeSort(right)
    if(left[length(left)] <= right[1]){
      return (c(left,right))
    }else{
      return (mergeTwo(left,right))
    }
  }
}
```

Proof of Concept

```
mergeSort(c(12,-22,13,2,-33,2))
```

```
## [1] -33 -22   2   2  12  13
```

RunTime

```
system.time(replicate(10, mergeSort(sample(x = 1:100, size = 10, replace = TRUE)))) / 10
```

```
##    user  system elapsed
##   3e-04   0e+00   3e-04
```

```
system.time(replicate(10, mergeSort(sample(x = 1:100, size = 100, replace = TRUE)))) / 10
```

```
##    user  system elapsed
##  0.0013  0.0000  0.0013
```

```
system.time(replicate(10, mergeSort(sample(x = 1:100, size = 1000, replace = TRUE)))) / 10
```

```
##    user  system elapsed
##  0.0252  0.0000  0.0252
```

```
system.time(replicate(10, mergeSort(sample(x = 1:100, size = 10000, replace = TRUE)))) / 10
```

```
##    user  system elapsed
##  1.1101  0.0001  1.1108
```

Quick Sort

Sorting Algorithm

```
quickSort <- function(vec){
  if(length(vec) > 1){
    pivot <- median(vec)
    return (c(quickSort(vec[vec < pivot]), vec[vec == pivot], quickSort(vec[vec > pivot])))
  }else{
    return (vec)
  }
}
```

Proof of Concept

```
quickSort(c(12,-22,13,2,-33,2))
```

```
## [1] -33 -22   2   2  12  13
```

RunTime

```
system.time(replicate(10, quickSort(sample(x = 1:100, size = 10, replace = TRUE)))) / 10
```

```
##    user  system elapsed
##   5e-04   0e+00   5e-04
```

```
system.time(replicate(10, quickSort(sample(x = 1:100, size = 100, replace = TRUE)))) / 10
```

```
##    user  system elapsed
##  0.0024  0.0000  0.0023
```

```
system.time(replicate(10, quickSort(sample(x = 1:100, size = 1000, replace = TRUE)))) / 10
```

```
##    user  system elapsed
##  0.0046  0.0000  0.0045
```

```
system.time(replicate(10, quickSort(sample(x = 1:100, size = 10000, replace = TRUE)))) / 10
```

```
##    user  system elapsed
##  0.0079  0.0000  0.0079
```