CT422 Development Project - Efficiency Analysis

All timings were done on a machine with: 2.4GHz intel i5 processor, with 4Gb RAM, running 64-bit Arch Linux(GNU/Linux). The c++ code was compiled using GCC 5.2.0. Using the supplied makefile, a standard version was compiled and also an optimised(-O3) version.

The tests were run using both versions. Runs were repeated 5 times for each version and the time taken was averaged. The test machine was also rebooted between each run to avoid any caching effects.

1 Stemmer

T_f - #Tokens in input file

- openfile() - $\mathcal{O}(T_f)$
  linearly scans over the input file (after profiling the program I believe this to be incorrect, the program spends the majority of it's execution time on line 28[tokenising]. I was unable to find the complexity of word_tokenize() or translate() but they must be quite expensive)

Average time taken to parse MED.ALL: 6.5305 seconds.
Average time taken to parse MED.QRY: 0.0424 seconds.

2 Indexer

member functions complexity:
k - constant, P_i - posting for term i, T_{d_i} - #Tokens in document i, N - #Docs in corpus

- addToIndex(d_i) - $\mathcal{O}\left(T_{d_i}\right)$
  linearly scans over the tokens in each doc
- getFreqInDoc(t_i) - $\mathcal{O}\left(|P_i|\right)$
  linearly scans over the posting for given term
- ComputeAvgDocLength() - $\mathcal{O}(N)$
  performs one constant time op. for each doc

- getAvgDocLength() - $\mathcal{O}(k)$
  avgDocLength is a member variable of the indexer (recomputed only when a document is added)
- getPostingFor() - $\mathcal{O}(k)$
- numDocsContaining() - $\mathcal{O}(k)$
- getDocStats() - $\mathcal{O}(k)$
- addTerm() - $\mathcal{O}(k)$
- addDocStats() - $\mathcal{O}(k)$

Regular: Average time taken to populate index: 328.8 milliseconds.
Optimised: Average time taken to populate index: 90.2 milliseconds.

3 Querying & Ranking

Q - Set of queries, T_Q - #Tokens in Q, q_i - single query, T_{q_i} - #Tokens in q_i, P_{t_i} - Posting for given term/token, C_R - Container holding the query rankings, C_Q - Container holding the queries

- RankGen() - $\mathcal{O}\left(T_Q * |P_{t_i}| * log(|C_R|)\right)$
  for each token in Q, linearly scan it's posting, update the relevant ranking
- getQueries() - $\mathcal{O}\left(T_Q * log(|C_Q|)\right)$
  performs one logarithmic time insertion for each token in Q
- queryCounter() - $\mathcal{O}\left(|Q|\right)$
  performs one constant time op. for each query

Regular: Average time taken to query index and rank results: 91.6 milliseconds.
Optimised: Average time taken to query index and rank results: 29.2 milliseconds.

4 Evaluating Perf.

Q - Query set, R - Rankings set($|Q| = |R|$), C_r - container holding relevance judgments, B_f - #Chars in relevance file

- evaluator() - $\mathcal{O}\left(|Q|^2 * log(|C_r|)\right)$
  for each query, for each ranking, check if there's a relevance judgement, do some arithmetic
- relSet() - $\mathcal{O}\left(B_f * log(|C_r|)\right)$
  iterates over relevance file by chars, inserts relevance judgments to std::set

Regular: Average time taken to evaluate rankings: 6.4 milliseconds.
Optimised: Average time taken to evaluate rankings: 2 milliseconds.

A Individual timings

stemmer	stemmer	regular	regular	regular	optimised	optimised	optimised
Docs	Queries	Indexer	RankGen	Evaluator	Indexer	RankGen	Evaluator
6.6262	0.0398	326	90	6	84	30	2
6.5903	0.0397	333	99	6	97	29	2
6.6353	0.0504	330	93	7	89	28	2
6.5922	0.0465	332	89	7	87	28	2
6.5724	0.0357	323	87	6	94	31	2
6.5303	0.0424	328.8	91.6	6.4	90.2	29.2	2

Python times in seconds, c++ time in milliseconds. (Couldn't get multi-column header rows working!)

B Complexity in STL containers

std::vector member functions complexity:

- push_back() - avg(amortized): constant, worst: linear
- pop_back() - constant
- size() - constant
- operator[] - constant
- erase() - linear in # elements erased

std::unordered_map member functions complexity:

- find() - avg: constant, worst: linear
- at() - avg: constant, worst: linear
- operator[] - avg: constant, worst: linear

std::map member functions complexity:

- insert() - logarithmic in size of map (in the form it's being used)
- operator[] - logarithmic in size of map

std::set member functions complexity:

- insert() - logarithmic in size of set (in the form it's being used)
- count() - logarithmic in size of set

std::multiset member functions complexity:

- insert() - logarithmic in size of multiset (in the form it's being used)
- count() - logarithmic in size of multiset and linear in # matches