CS2040S 22/23 Sem 1

Orders of Growth

$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < 2^n < 2^{2n}$
$\log_a n < n^a < a^n < n! < n^n$

Types

properties

Let $T(n) = O(f(n))$ and $S(n) = O(g(n))$
- addition: $T(n) + S(n) = O(f(n) + g(n))$
- multiplication: $T(n) * S(n) = O(f(n) * g(n))$
- composition: $f_1 \circ f_2 = O(g_1 \circ g_2)$
  - only if both functions are increasing

1 HASH TABLES

- disadvantage: no successor/predecessor operation

1.1 hashing

Let the m be the table size; let n be the number of items; let $cost(h)$ be the cost of the hash function
- $load$(hash table), $\alpha = \frac{n}{m}$ = average number of items per bucket = expected number of items per bucket

1.1.1 hashing assumptions

- **simple uniform hashing assumption**
  - every key has an equal probability of being mapped to every bucket
  - keys are mapped independently
- **uniform hashing assumption**
  - every key is equally likely to be mapped to every permutation, independent of every other key.
  - NOT fulfilled by linear probing

1.1.2 properties of a good hash function

1. able to enumerate all possible buckets - $h : U \to \{1..m\}$
   - for every bucket j, $\exists i$ such that $h(key, i) = j$
2. simple uniform hashing assumption

1.2 chaining

- time complexity
  - `insert(key, value)` - $O(1 + cost(h)) \implies O(1)$
    - for n items: expected maximum cost
      - $= O(\log n)$
      - $= \Theta(\frac{\log n}{\log(\log(n))})$
  - `search(key)`
    - worst case: $O(n + cost(h)) \implies O(n)$
    - expected case: $O(\frac{n}{m} + cost(h)) \implies O(1)$
- total space: $O(m + n)$

1.3 open addressing - linear probing

- $hash(k) = (k\%m + i)\%TABLE\_SIZE$
  (if collide, check next slot)
- `delete(key)`
  - use a **tombstone value** - DON'T set to `null`
- **performance**
  - if the table is $\frac{1}{4}$ full, there will be clusters of size $\Theta(\log n)$
  - expected cost of an operation i.e no. of probes $\leq \frac{1}{1-\alpha}$ (assume $\alpha < 1$ and uniform hashing)
- **advantages**
  - saves space (use empty slots vs linked list)
  - better cache performance (table is one place in memory)
  - rarely allocate memory (no new list-node allocation)
- **disadvantages**
  - more sensitive to choice of hash function (primary clustering)
  - more sensitive to load (as $\alpha \to 1$, performance degrades)

1.3.1 modified linear probing

```
hash(key)
(hash(key) + 1 * d) % m
(hash(key) + 2 * d) % m
(hash(key) + 3 * d) % m
```

d is some constant integer > 1 and is co-prime to m

1.3.2 quadratic probing

```
hash(key)
(hash(key) + 1) % m
(hash(key) + 4) % m
(hash(key) + 9) % m
:
(hash(key) + k^2) % m
```

- If $\alpha < 0.5$ and m is prime, then we can always find an empty slot.

1.3.3 double hashing

$(hash(key) + i * hash_2(key))\%TABLE\_SIZE$
- **Secondary hash function must not evaluate to 0**
- To solve this problem, simply change $hash_2(key)$ to: $hash_2(key) = n - (key\%n)$

Prevents secondary clustering

sort	**best**	**average**	**worst**	**stable?**	**in-place**
bubble	$\Omega(n)$	$O(n^2)$	$O(n^2)$	✓	✓
radix	$\Omega(n)$	$O(n)$	$O(n)$	✓	×
selection	$\Omega(n^2)$	$O(n^2)$	$O(n^2)$	×	✓
insertion	$\Omega(n)$	$O(n^2)$	$O(n^2)$	✓	✓
merge	$\Omega(n \log n)$	$O(n \log n)$	$O(n \log n)$	✓	×
quick	$\Omega(n \log n)$	$O(n \log n)$	$O(n^2)$	×	✓
heap	$\Omega(n \log n)$	$O(n \log n)$	$O(n \log n)$	×	×

sorting invariants

sort	**invariant** (after k iterations)
bubble	largest k elements are sorted
selection	smallest k elements are sorted
insertion	first k slots are sorted
merge	given subarray is sorted
quick	partition is in the right position

searching

search	**average**
linear	$O(n)$
binary	$O(\log n)$
quickSelect	$O(n)$

data structures assuming $O(1)$ comparison cost

data structure	**search**	**insert**
sorted array	$O(\log n)$	$O(n)$
unsorted array	$O(n)$	$O(1)$
linked list	$O(n)$	$O(1)$
tree (kd/(a, b)/binary)	$O(\log n)$ or $O(h)$	$O(\log n)$ or $O(h)$
trie	$O(L)$	$O(L)$
symbol table	$O(1)$	$O(1)$
chaining	$O(n)$	$O(1)$
open addressing	$\frac{1}{1-\alpha} = O(1)$	$O(1)$

orders of growth

$$T(n) = 2T(\frac{n}{2}) + O(n) \Rightarrow O(n \log n)$$
$$T(n) = T(\frac{n}{2}) + O(n) \Rightarrow O(n)$$
$$T(n) = 2T(\frac{n}{2}) + O(1) \Rightarrow O(n)$$
$$T(n) = T(\frac{n}{2}) + O(1) \Rightarrow O(\log n)$$
$$T(n) = 2T(n-1) + O(1) \Rightarrow O(2^n)$$
$$T(n) = 2T(\frac{n}{2}) + O(n \log n) \Rightarrow O(n(\log n)^2)$$
$$T(n) = 2T(\frac{n}{4}) + O(1) \Rightarrow O(\sqrt{n})$$
$$T(n) = T(n-c) + O(n) \Rightarrow O(n^2)$$