1. (20 Punkte) Eine DEQUEUE ist eine Datenstruktur, die eine Folge $X_i, X_{i+1}, \ldots, X_j$ von Stücken speichert, mit i, j ganze Zahlen. Folgende Operationen sind darauf definiert:

   - Q.SIZE() gibt die Länge der Folge Q an
   - Q.EMPTY() gibt an, ob die Folge Q leer ist
   - Q.FIRST() gibt das erste Stück in der Folge Q zurück (also X_i)
   - Q.LAST() gibt das letzte Stück in der Folge Q zurück (also X_j)
   - Q.ENDPUSH(X) hängt das Stück x ans Ende der Folge Q an (sozusagen als X_{j+1})
   - Q.FRONTPUSH(X) hängt das Stück x an den Anfang der Folge Q (sozusagen als X_{i-1})
   - Q.ENDPOP() entfernt das letzte Stück von Q und gibt es zurück
   - Q.FRONTPOP() entfernt das erste Stück von Q und gibt es zurück

   Die beiden Pop-Operationen verursachen einen Fehler, wenn Q leer ist. Man kann also eine Dequeue als eine Art doppelendigen Stack betrachten.

   Geben Sie eine Implementierung dieser Datenstruktur an, die auf Arrays basiert, die jede Operation in konstanter amortisierter Laufzeit realisiert, und die nicht übermäßig Speicherplatz verschwendet.

   Erklären Sie, warum Sie konstante amortisierte Laufzeit erreichen.

2. (10 Punkte) Eine sortiertes Feld erlaubt das Suchen in logarithmischer Zeit (binäre Suche), aber das Einfügen und Löschen braucht im schlechtesten Fall lineare Zeit.

   Betrachten wir den Fall, in dem nur gesucht und gelöscht wird. Also man beginnt mit einem sortierten Feld und möchte dann eine Folge von Such- und Löschoperationen durchführen. Um das teure Verschieben von Teilfeldern zu sparen, kann man auch gelöschte Elemente als solche markieren und sie dann in folgenden Suchoperationen ignorieren. Damit würde das Löschen eines Stücks, nachdem es gefunden wurde, nur mehr konstante Zeit brauchen.

   (a) Wie würde das alles genau funktionieren? Versuchen Sie das in einer imperativen Programmiersprache Ihrer Wahl zu spezifizieren. Nehmen Sie an, dass alle Schlüssel verschieden sind.

   (b) Wie geht man mit dem Fall um, dass gleiche Schlüssel erlaubt sind? Es ist sinnvoll, sich für diesen Fall zuerst die tatsächliche Problemstellung genau zu überlegen.

3. (15 Punkte) Erweitern Sie Power$_{\text{of 2}}$dictionaries, so dass auch Löschoperationen möglich sind. Welche amortisierte Laufzeiten können Sie für Suche, Einfügen, Löschen erzielen? Wie sieht es aus, wenn Sie für das Löschen eines Elements die Zeit fürs Finden nicht miteinbeziehen müssen?

   Hinweis: Bei der Analyse kann es hilfreich sein, separate Konten fürs Einfügen und Löschen zu verwenden.