

アルゴリズムとデータ構造

第13週目

担当 情報システム部門 徳光政弘
2025年7月23日

今日の内容

- ヒープソート
- 計算量の比較
- ソートの安定性

ヒープ

- データに優先度を持たせて保持するデータ構造および手順
- 病院の順番待ち
 - 基本は受付順
 - 急患が来れば、急患を優先する

ヒープのデータ操作

- ヒープをHとする
- プッシュヒープ `push_heap(H, x)`
 - xを追加する
- デリートマキシマム `delete_maximum(H, x)`
 - 最大値を削除して、取り出す

ヒープの実現方法

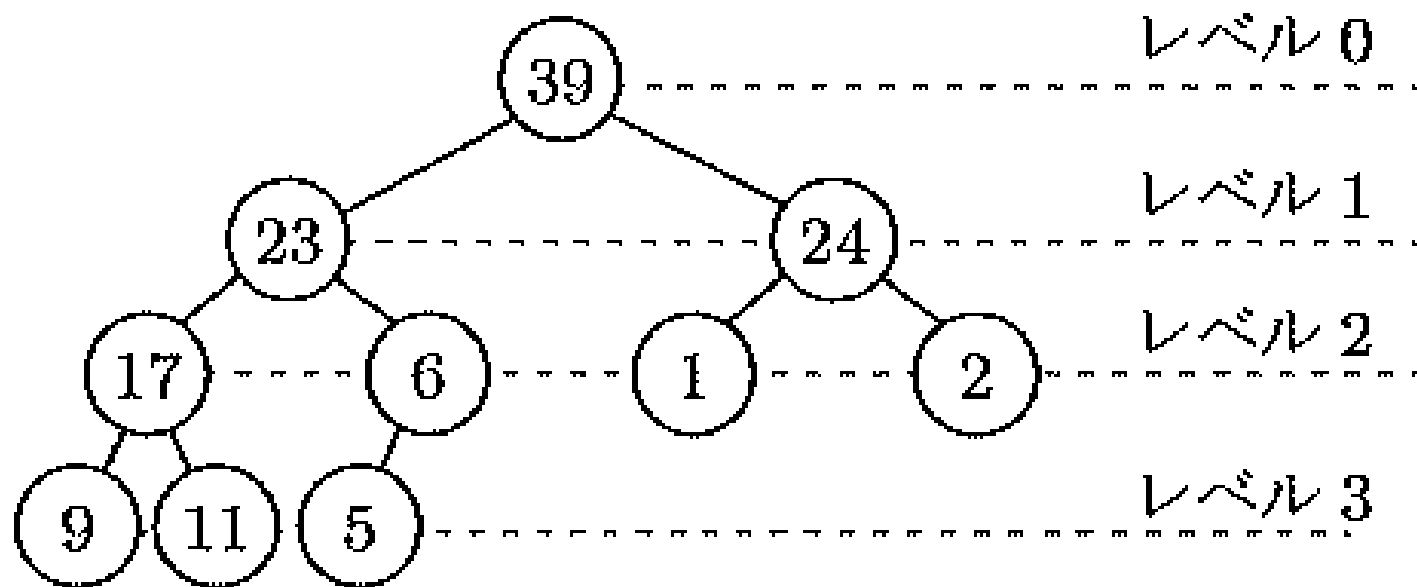


図 5.6 2 分木によるヒープ

ヒープに必要な性質

◆定義 5.3 ヒープ

以下の2つの性質が成り立つ2分木をヒープとよぶ.

性質1 2分木の最大のレベルを l_m とすると, $0 \leq k \leq l_m - 1$ を満たす各レベル k には 2^k 個の節点が存在し, レベル l_m に存在する葉はそのレベルに左詰めされている.

性質2 各節点に保存されるデータは, その子に保存されるデータより大きい.

ヒープの実現方法(性質1)

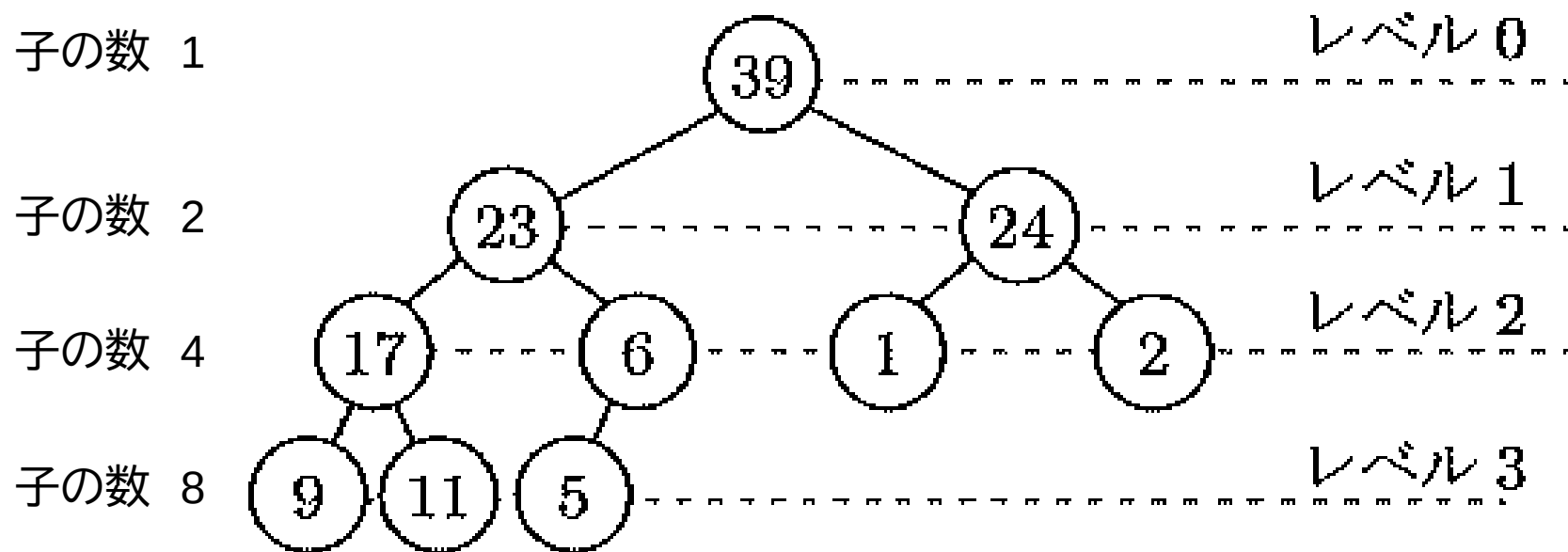


図 5.6 2 分木によるヒープ

ヒープの実現方法(性質2)

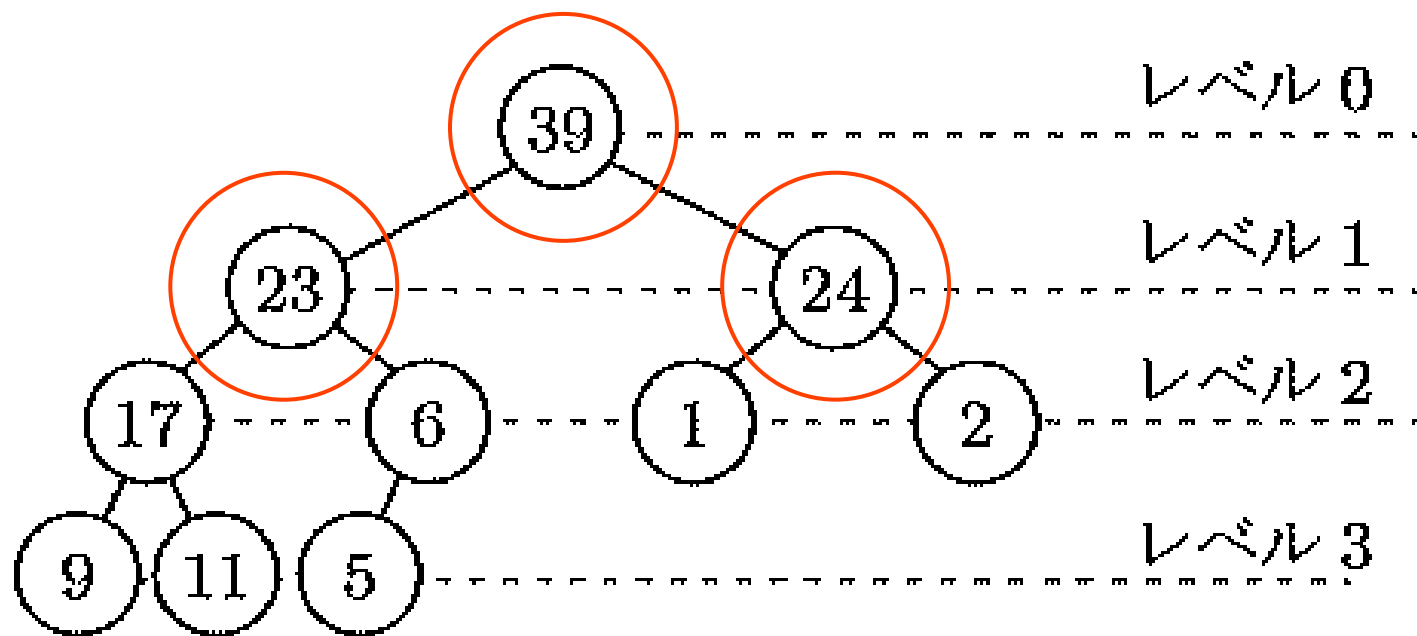


図 5.6 2 分木によるヒープ

ヒープへのデータ追加

- ① データ x を格納する節点を作成し、その節点を定義 5.3 の性質 1 を満たすような葉として追加する.
- ② 追加したデータ x を含む節点と、その節点の親節点のデータを比較する.
 - 親節点のデータが大きければ、定義 5.3 の性質 2 を満たしているので、格納操作を終了する.
 - 親節点のデータが小さければ、親子の節点間のデータを交換し、②の操作を根に向かって繰り返す.

ヒープへのデータ追加

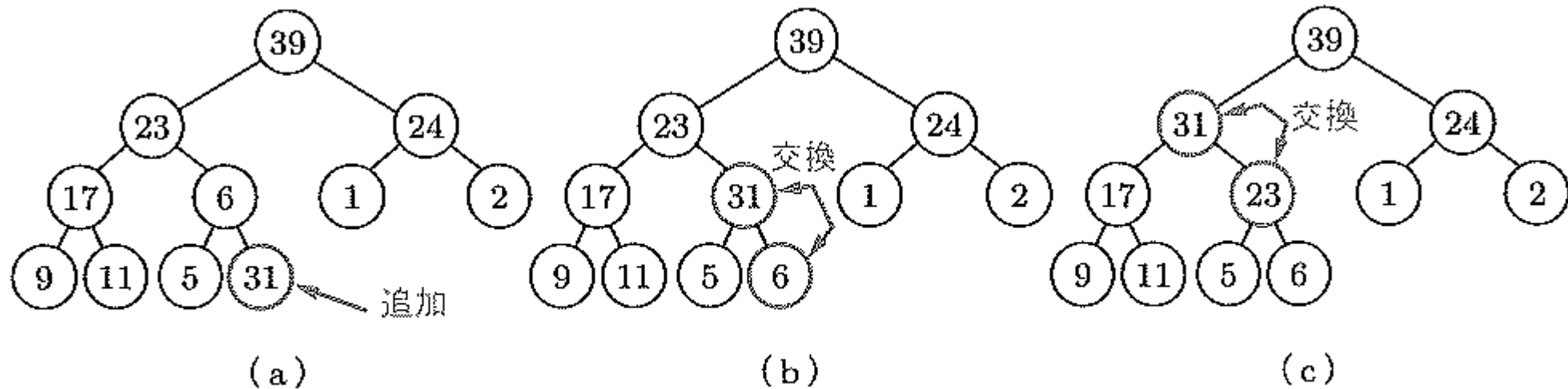


図 5.7 ヒープへのデータの追加

追加したデータを並べ替えていく

ヒープへのデータ追加

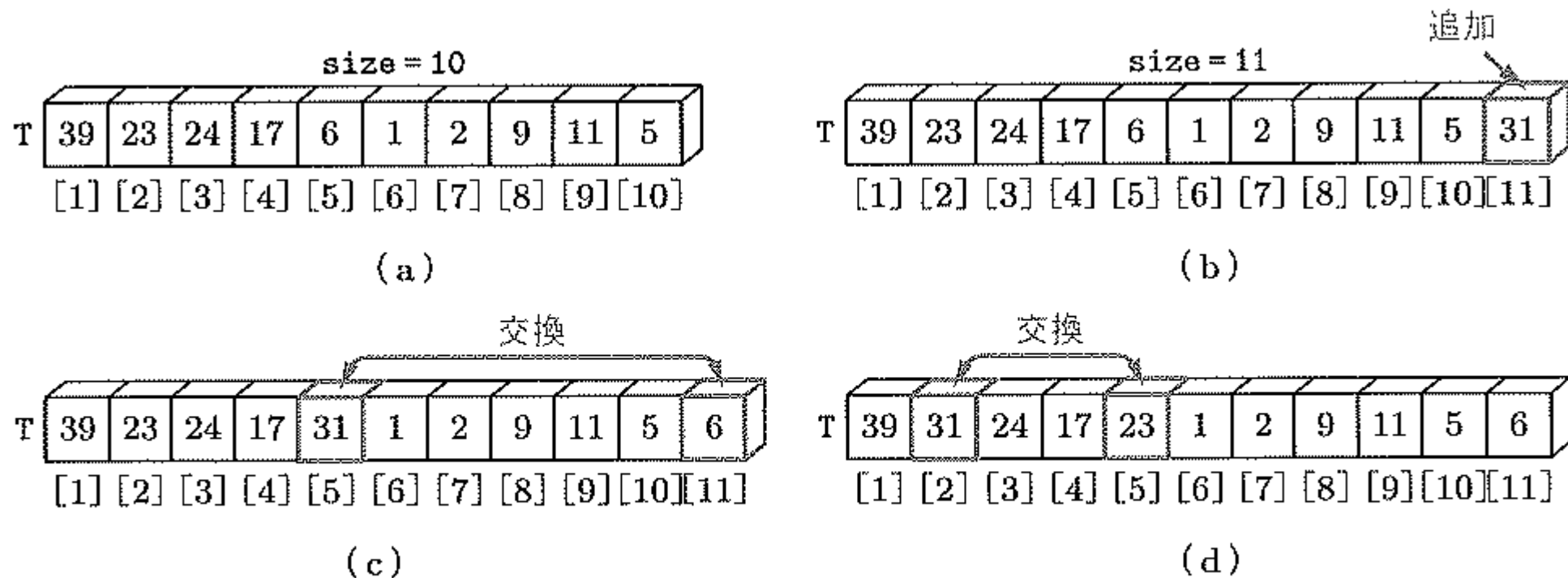



図 5.8 ヒープを表す配列に対するデータの追加
2分木を実現する添字の計算になっている点に注意

ヒープへのデータ追加

アルゴリズム 5.3 関数 `push_heap`

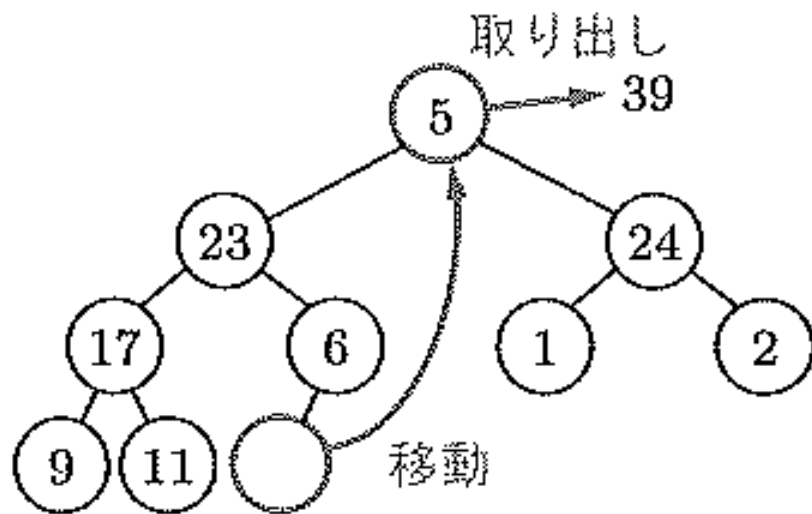
入力：ヒープを表す配列 $T[1], T[2], \dots, T[n]$ と追加される値 x

```
push_heap(T,x) {  
    size=size+1;  
    T[size]=x;           //データを最後に追加  
    k=size;  
    while ((T[k]>T[k/2])かつ(k>1)) { //親の値と比較  
        swap(T[k],T[k/2]);          //親の値が小さければ値を交換  
        k=k/2;  根の方向への添字を計算  
    }  
}
```

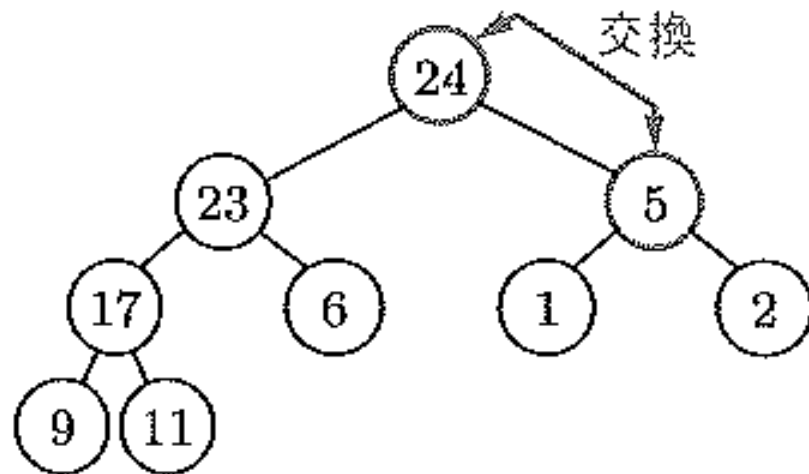
ヒープからのデータの取り出し(ソート)

- 根に最大値が保存されている
- 最大値をとりだした後は並べ替え(根の最大値)が必要

ヒープからのデータの取り出し(ソート)



(a)



(b)

図 5.9 ヒープからの最大値の取り出し

ヒープからのデータの取り出し(ソート)

- ① 根から最大値を取り出し、そこに右端の葉のデータを格納する(右端の葉は削除する).
- ② 根に移動したデータを含む節点と、その節点の子節点のデータを比較し、比較結果に従って以下の処理を実行する.

子節点が存在しない場合：操作を終了する.

子節点が1つの場合：子節点のデータが小さければ、性質2を満たしているので操作を終了し、子節点のデータが大きければ、2つの節点のデータを交換し、②の操作を葉に向かって繰り返す.

子節点が2つの場合：2つの子節点のデータが両方とも根に移動したデータより小さければ、性質2を満たしているので操作を終了する. いずれかの子節点のデータが大きければ、根に移動したデータと大きいほうの値をもつ子節点のデータを交換し、②の操作を葉に向かって繰り返す.

ヒープからのデータの取り出し(ソート)

アルゴリズム 5.4 関数 `delete_maximum`

入力：ヒープを表す配列 $T[1], T[2], \dots, T[n]$

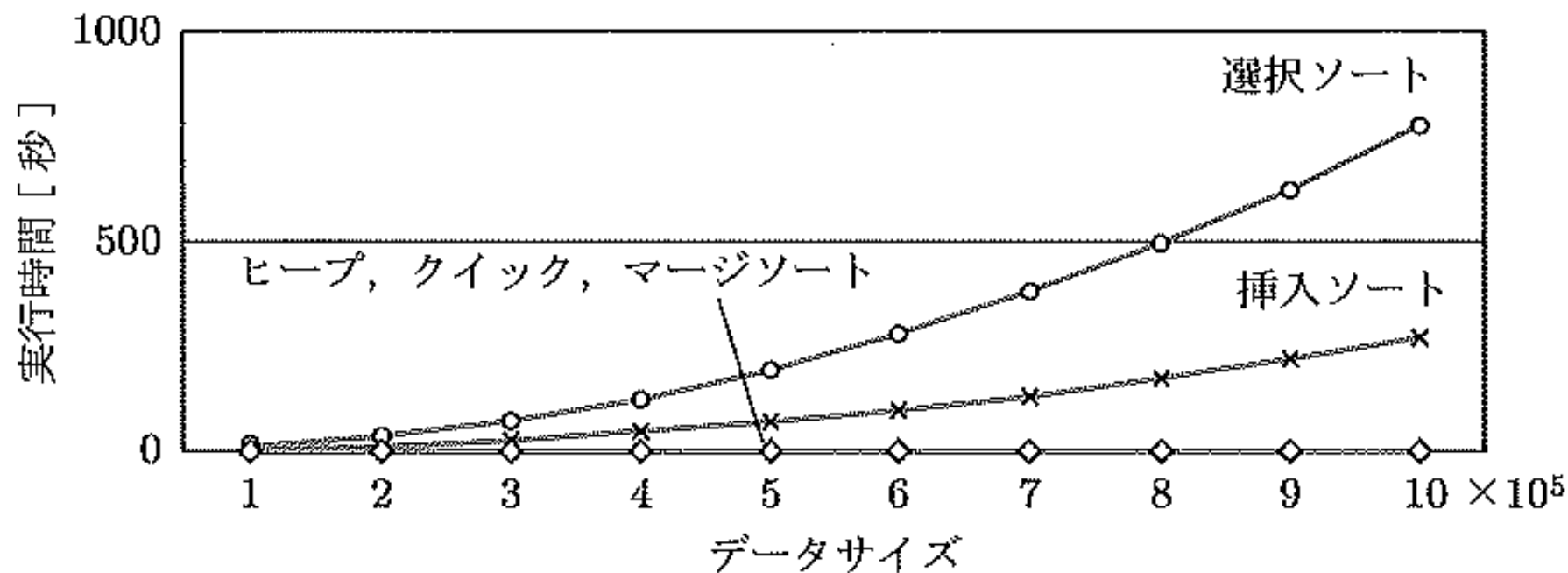
```
delete_maximum(T) {  
    T[1]を出力;  
    T[1]=T[size]; T[size]を空にする;    //葉のデータを根に移動  
    size=size-1; k=1;  
    while (2*k<=size) {                  //子をもつかどうかを判定  
        if (2*k==size) {                 //子が1つの場合  
            if (T[k]<T[2*k]) {             //親子の値を比較  
                swap(T[k],T[2*k]); k=2*k; //親の値が小さい場合は値を交換  
            }  
        }  
        else { アルゴリズムを終了; }  
    }  
}
```


時間計算量の考え方

- push_heapとdelete_maximumはそれぞれの計算量が $O(n \log(n))$ になっている

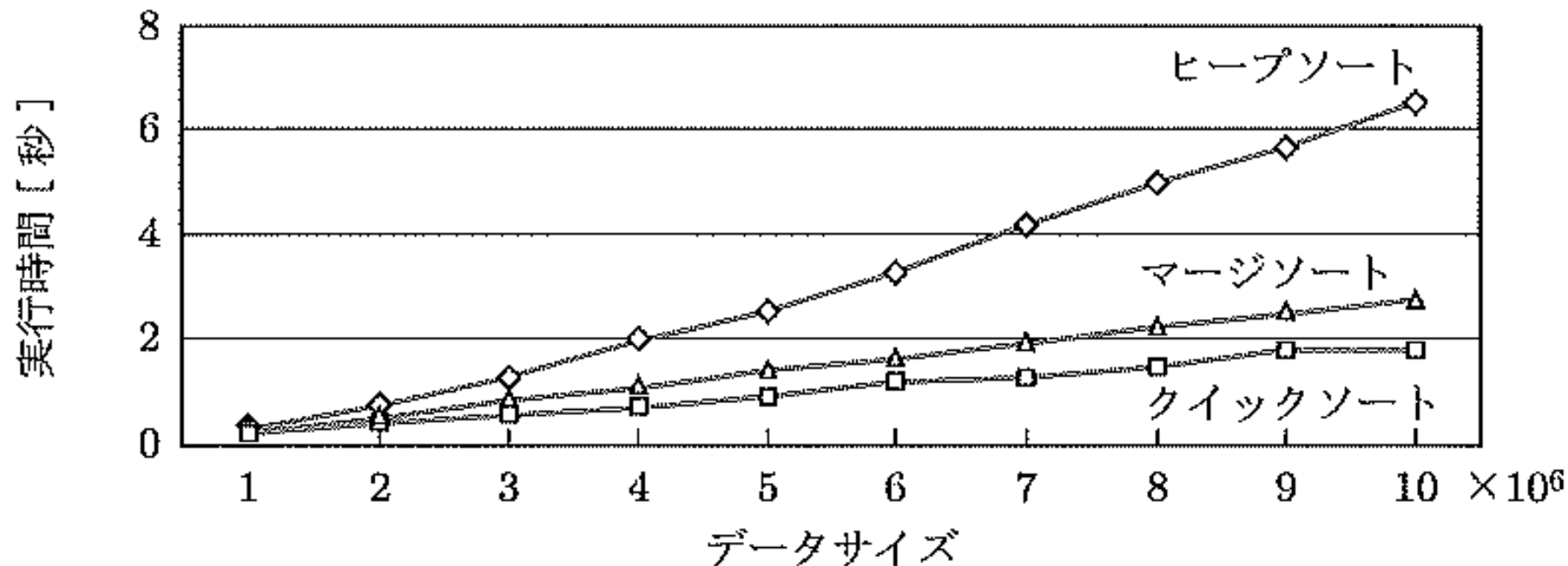
$$2 \times \sum_{i=1}^n \log i \leq 2 \times n \times \log n = O(n \log n)$$

ソートの性能比較



(a) 5つのソートアルゴリズムの比較

ソートの性能比較



(b) ヒープソート，クイックソートとマージソートの比較

ソートの安定性

- 同じ値がある場合にどのように取り扱うか
- 比較できるようにして並べ替える

学籍番号	氏名	点数
3001	石川	90
3002	川上	80
3003	中村	90
3004	深川	90
3005	野中	70

(a)

学籍番号	氏名	点数
3005	野中	70
3002	川上	80
3003	中村	90
3004	深川	90
3001	石川	90

(b)

学籍番号	氏名	点数
3005	野中	70
3002	川上	80
3001	石川	90
3003	中村	90
3004	深川	90

(c)

図 6.8 同じ値のデータをもつ入力のソート

ソートの安定性と必要な性質

- データの値の大きさを順番に並べる
- データが与えられた順番も本来は考慮が必要

学籍番号	氏名	点数
3001	石川	90
3002	川上	80
3003	中村	90
3004	深川	90
3005	野中	70

(a)

学籍番号	氏名	点数
3005	野中	70
3002	川上	80
3003	中村	90
3004	深川	90
3001	石川	90

(b)

学籍番号	氏名	点数
3005	野中	70
3002	川上	80
3001	石川	90
3003	中村	90
3004	深川	90

(c)

図 6.8 同じ値のデータをもつ入力のソート

ソートの安定性と必要な性質

- クイックソートは不安定
- 分割して交換する過程で、基準データに対して、同じ値が複数ある場合に、partition操作の結果、後ろのデータが前に配置される可能性がある

クイックソートは不安定(unstable)ですよという端的な例 |
GEO Solutions 技術情報

<https://geo-sol.co.jp/tech/it/20160801/>

