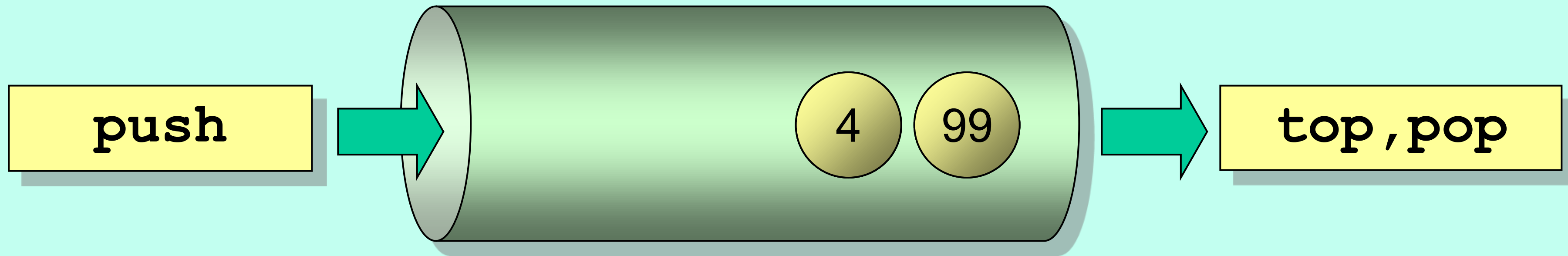
Binary Heap

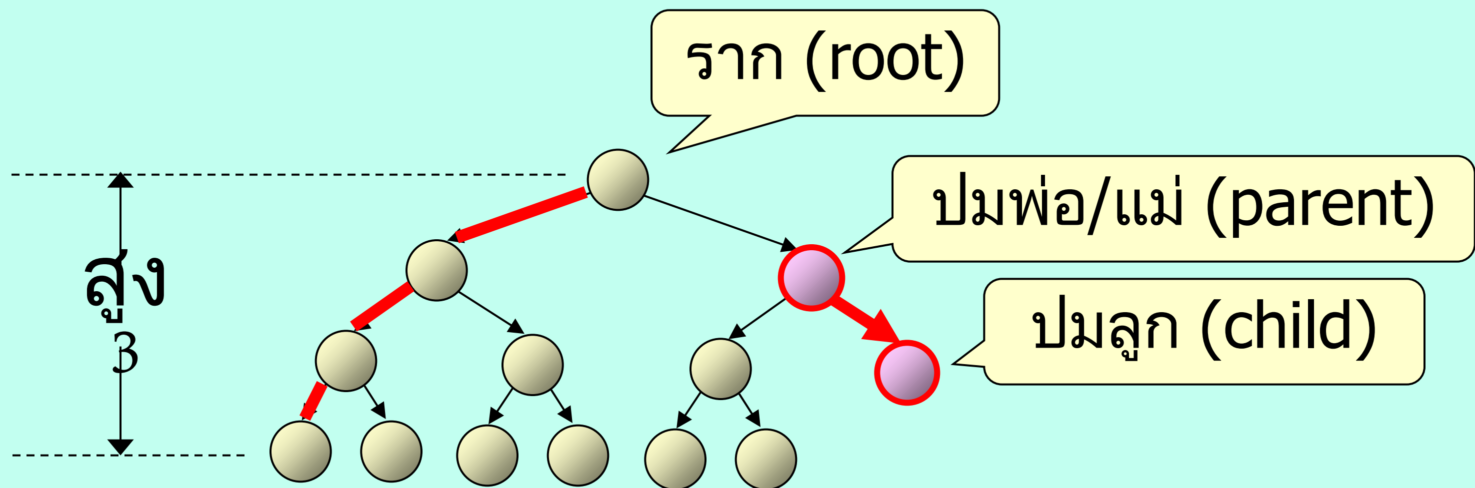
สมชาย ประสิทธิ์จูตระกูล

priority_queue



สร้าง Priority Queue ด้วย Binary Heap

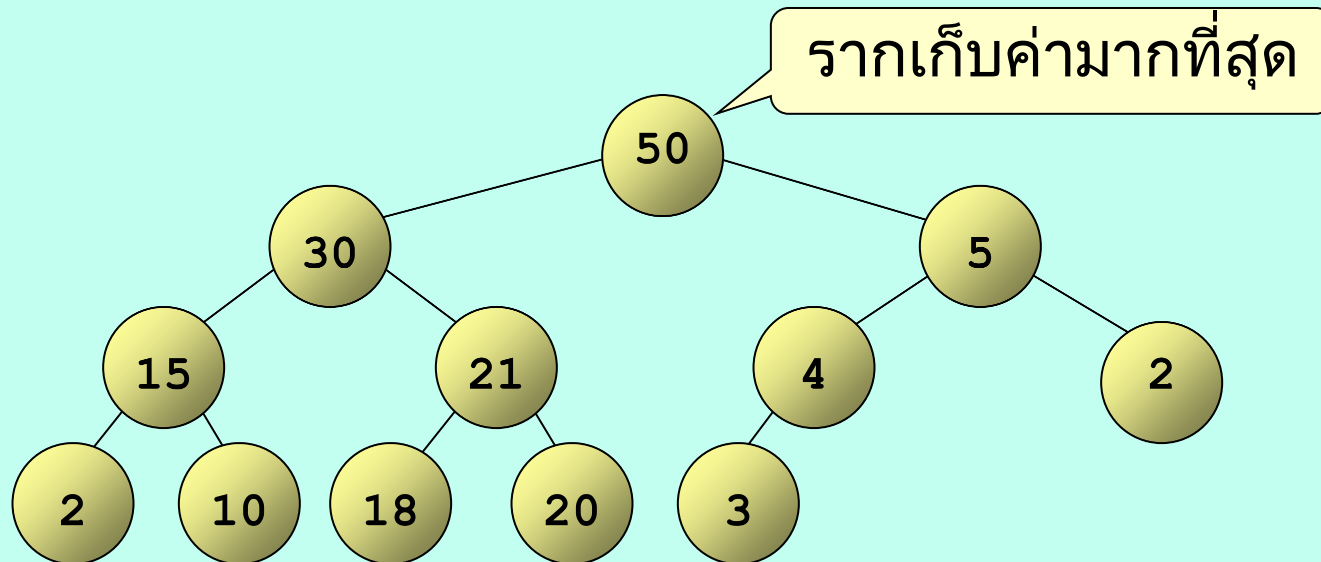
- push : O(log n)
- pop : O(log n)
- top : Θ(1)
- ใช้โครงสร้างแบบ balanced binary tree



ต้นไม้แบบทวิภาคได้ดุลที่มี n ปม สูง $\lfloor \log_2 n \rfloor$

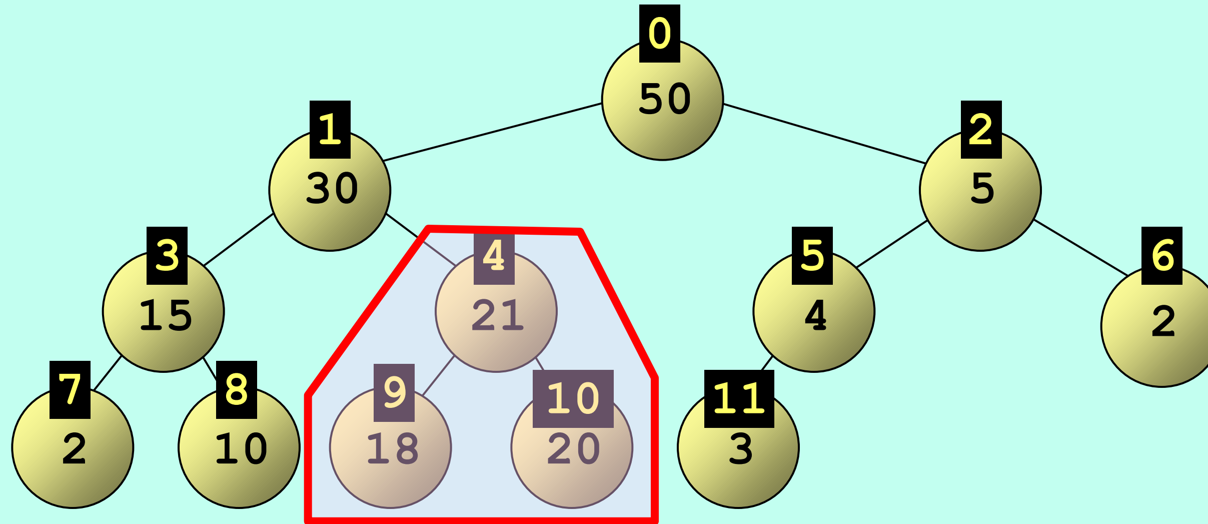
ฮีปแบบทวิภาค

- Binary Heap
  - มีโครงสร้างเป็น binary tree แบบได้ดุล
    - node เต็มทุกระดับ
    - ระดับล่างสุด เต็มจากซ้ายไปขวา
  - ข้อมูลของ parent node มีค่ามากกว่าของลูก ๆ



© S. Prasitjutrakul 2013

การสร้างฮีปแบบทวิภาคด้วยอาเรย์



mSize
12

0	1	2	3	4	5	6	7	8	9	10	11	12	13
50	30	5	15	21	4	2	2	10	18	20	3		

mSize **mData**

- รากเก็บที่ index 0
- ลูก<u>ซ้าย</u>ของ node ที่ index k อยู่ที่ index 2k + 1
- ลูก<u>ขวา</u>ของ node ที่ index k อยู่ที่ index 2k + 2
- พ่อของ node ที่ index k อยู่ที่ index (k – 1) / 2

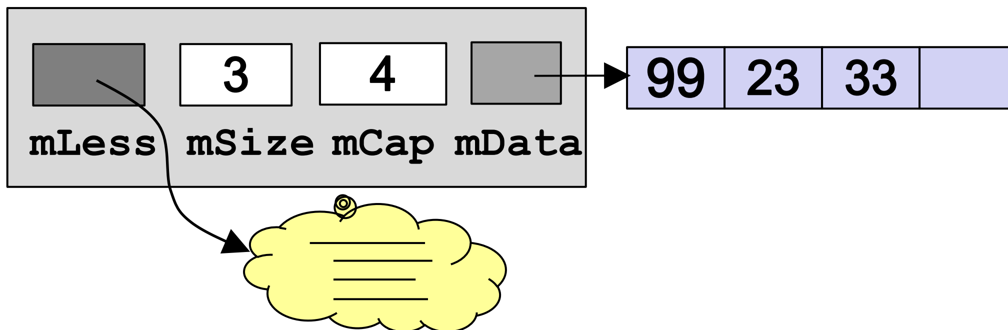
priority_queue : บริการ

```
priority_queue<T>
priority_queue<T, Comp>

bool          empty();
size_t        size();
const T&      top();
void          push(const T& e);
void          pop();
```

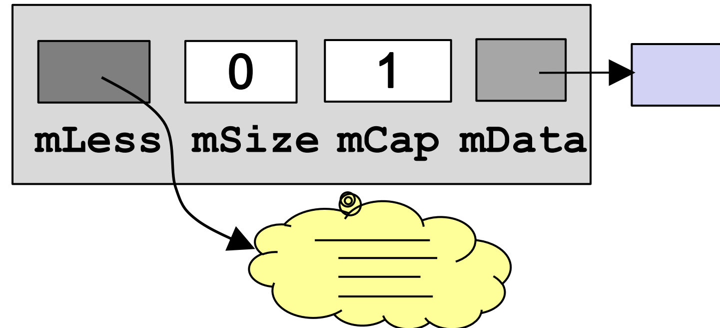
```
priority_queue<T,Comp>&
              operator=(priority_queue<T,Comp> rhs)
```

คลาส priority_queue

```
template <typename T, typename Comp = std::less<T>>
class priority_queue {
  protected:
    T*     mData;
    size_t mCap;
    size_t mSize;
    Comp   mLess;

  public:
    priority_queue(const Comp& c = Comp())                { ... }
    priority_queue(const priority_queue<T,Comp>& a)  { ... }
    ~priority_queue()                                         { ... }

    //------------- capacity function ----------
    bool   empty() const           { ... }
    size_t size()  const           { ... }
    //----------------- access -----------------
    const T& top() const           { ... }
    //----------------- modifier ---------------
    void push(const T& e)          { ... }
    void pop()                     { ... }
};
```



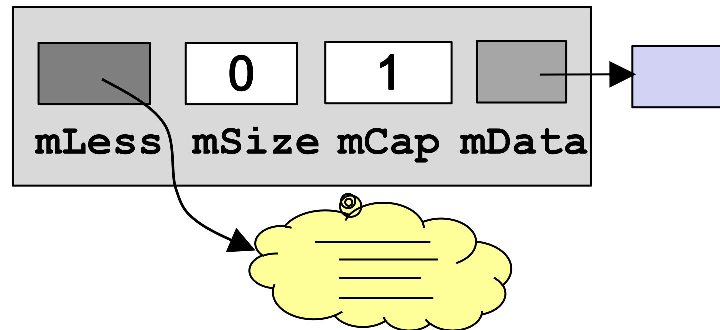
```
template <typename T, typename Comp = std::less<T> >
class priority_queue {
  protected:
    T*     mData;
    size_t mCap;
    size_t mSize;
    Comp   mLess;
  public:
    priority_queue(const Comp& c = Comp()) {
      mCap  = 1;
      mData = new T[mcap]();
      mSize = 0;
      mLess = c;
    }
    priority_queue(const priority_queue<T,Comp>& a) {
      mCap  = a.mCap;
      mData = new T[mcap]();
      for (size_t i=0; i<a.mCap; i++) mData[i] = a.mData[i];
      mSize = a.mSize;
      mLess = a.mLess;
    }
    ~priority_queue() {
       delete [] mData;
    }
```



©

constructor : initialization list

```
template <typename T, typename Comp = std::less<T> >
class priority_queue {
  protected:
    T*      mData;
    size_t mCap;
    size_t mSize;
    Comp   mLess;

  public:
    priority_queue(const Comp& c = Comp()) :
                  mData(new T[1]()), mCap(1),
                  mSize(0), mLess(c) {
    }
    priority_queue(const priority_queue<T,Comp>& a) :
                  mData(new T[a.mCap]()), mCap(a.mCap),
                  mSize(a.mSize), mLess(a.mLess) {
      for (size_t i=0; i<a.mCap; i++) mData[i] = a.mData[i];
    }
    ~priority_queue() {
       delete [] mData;
    }
    ...
```



©

c++ : initialization list

```
class A {
  public:
    A() {
      a0 = 2; a1 = 3;
    }
    A(int x0, int x1) {
      a0 = x0; a1 = x1;
    }
  protected:
    int a0, a1;
};
class B : public A {
  public:
    B() { b = a0+a1; }
  public:
    int b;
};

int main() {
    B b;
    cout << b.b << endl; //5
}
```

```
class A {
  public:
    A() :  a0(2), a1(3) {

    }
    A(int x0, int x1) :
      a0(x0), a1(x1) {
    }
  protected:
    int a0, a1;
};
class B : public A {
  public:
    B() : A(), b(a0+a1) {}
  public:
    int b;
};

int main() {
    B b;
    cout << b.b << endl; //5
}
```

c++ : initialization list

```
class A {
  public:
    A() : a0(2), a1(3) { }
    A(int x0, int x1) :
       a0(x0), a1(x1) { }
  protected:
    int a0, a1;
};
class B : public A {
  public:
    B() : A(), b(a0+a1) {}
  public:
    int b;
};

int main() {
    B b;
    cout << b.b << endl; //5
}
```

```
class A {
  public:
    A() : a0(2), a1(3) { }
    A(int x0, int x1) :
       a0(x0), a1(x1) { }
  protected:
    int a0, a1;
};
class B : public A {
  public:
    B() : A(4,5), b(a0+a1) {}
  public:
    int b;
};

int main() {
    B b;
    cout << b.b << endl; //9
}
```

c++ : initialization list

```
class A {
  public:
    A() : a0(2), a1(3) { }
    A(int x0, int x1) :
       a0(x0), a1(x1) { }
  protected:
    int a0, a1;
};
class B : public A {
  public:
    B() : A(4,5), b(a0+a1) {}
  public:
    int b;
};

int main() {
    B b;
    cout << b.b << endl; //9
}
```

```
class A {
  public:
    A() : a0(2), a1(3) { }
    A(int x0, int x1) :
       a0(x0), a1(x1) { }
  protected:
    int a0, a1;
};
class B : public A {
  public:
    B() {
      A(4,5);  // <-- สร้าง obj
      b = a0 + a1; // 2+3
    }
  public:
    int b;
};

int main() {
    B b;
    cout << b.b << endl; //5
}
```

© S. Prasitjutrakul 2013

c++ : initialization list

- เร็วกว่า (ไม่ต้องตั้งค่า data members หลายครั้ง)
- เรียก non-default ctor ของ parent class ได้
- เรียก non-default ctor ของ member ได้
- ใช้ตั้งค่าให้ const และ reference members

```
class A {
  const int a;
  public:
    A()        : a(2) { }
    A(int a) : a(a) { }

};
```

```
class A {
  const int a;
  public:
    A() { a = 2; } //ผิด
};
```

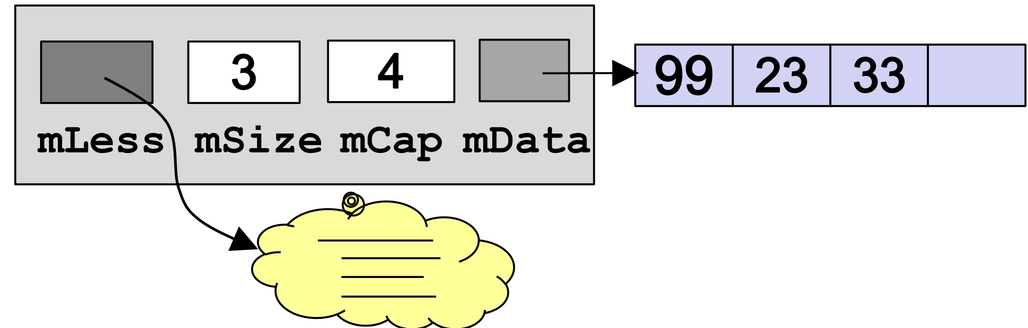
```
class A {
  int & a;
  public:
    A(int a) : a(a) { }
    ...
};
int main() {
  int x = 3;
  A a(x);
  x = 2;
  ...
```

```
class A {
  Foo f;
  public:
    A()        : f(0) { }
    A(int a) : f(a) { }
};
```

© S. Prasitjutrakul 2013

copy assignment operator : วิธีที่ 1

```
template <typename T, typename Comp = std::less<T> >
class priority_queue {
  protected:
    T*     mData;
    size_t mCap;
    size_t mSize;
    Comp   mLess;

  public:
    ...
    priority_queue<T,Comp>& operator=
                          (const priority_queue<T,Comp>& rhs) {
      using std::swap;
      priority_queue<T,Comp> temp(rhs);
      swap(mSize, temp.mSize);
      swap(mCap,  temp.mCap );
      swap(mData, temp.mData);
      swap(mLess, temp.mLess);
      return *this;
    }
    ...
```



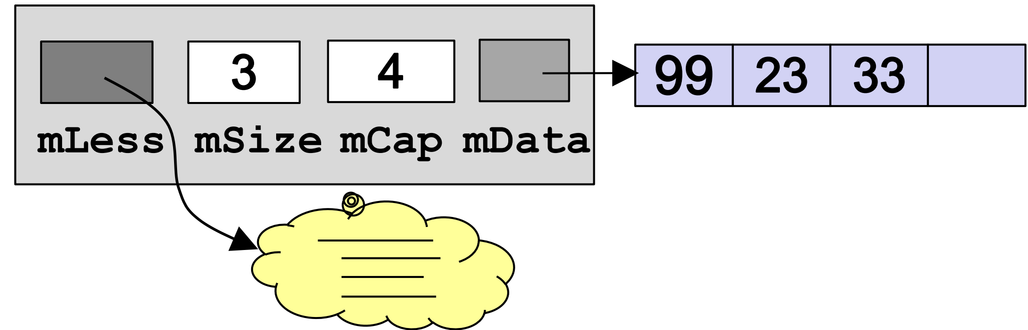
pass by reference

```
priority_queue<int> pq1, pq2;
for (int i=0; i<10; i++) pq1.push(1);
pq2 = pq1;
```

© S. Prasitjutrakul 2013

copy assignment operator : วิธีที่ 2

```
template <typename T, typename Comp = std::less<T> >
class queue {
  protected:
    T*     mData;
    size_t mCap;
    size_t mSize;
    Comp   mLess;

  public:
    ...
    priority_queue<T,Comp>& operator=(priority_queue<T,Comp> rhs){
      using std::swap;
      swap(mSize, rhs.mSize);
      swap(mCap,  rhs.mCap );
      swap(mData, rhs.mData);
      swap(mLess, rhs.mLess);
      return *this;
    }
    ...
```



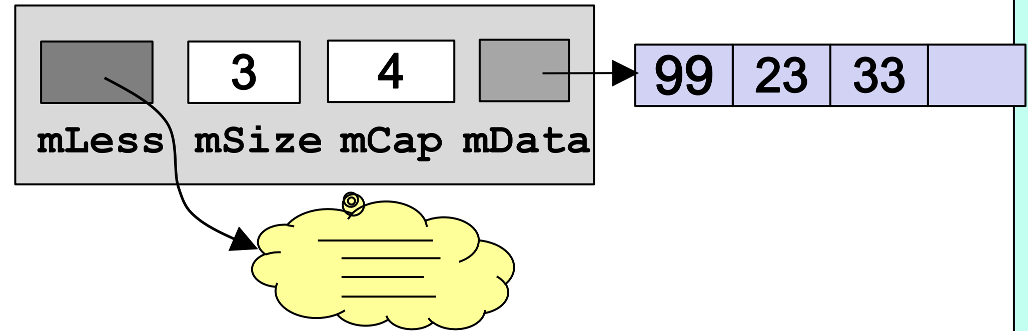
pass by value

```
priority_queue<int> pq1, pq2;
for (int i=0; i<10; i++) pq1.push(1);
pq2 = pq1;
// below is copy initialization
priority_queue<int> pq3 = pq1; // !!!
```

© S. Prasitjutrakul 2013

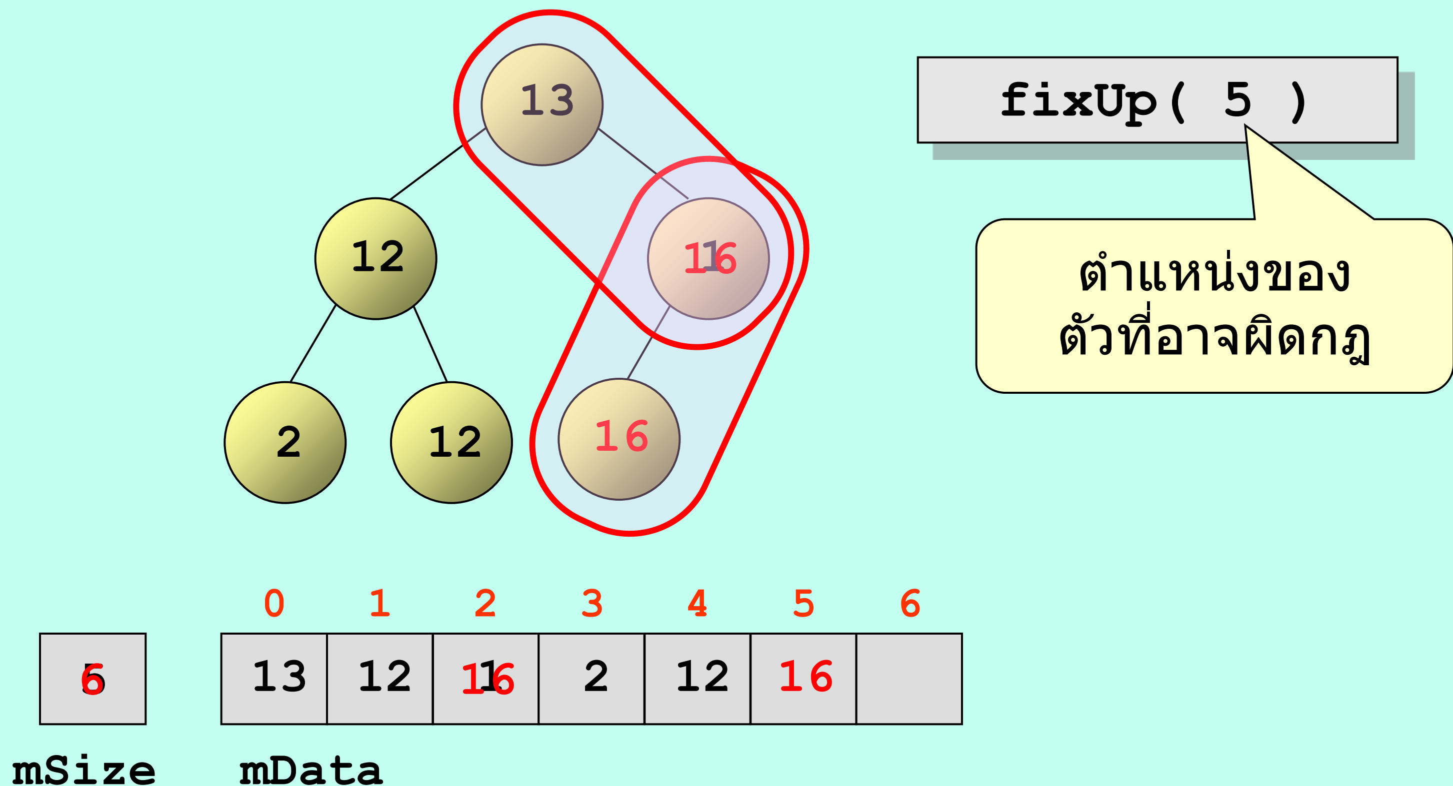
empty, size, top

```
template <typename T, typename Comp = std::less<T> >
class priority_queue {
  protected:
    T*     mData;
    size_t mCap;
    size_t mSize;
    Comp   mLess;

  public:
    ...
    bool empty() const {     // Θ(1)
      return mSize == 0;
    }

    size_t size() const {   // Θ(1)
      return mSize;
    }

    const T& top() {         // Θ(1)
      return mData[0];
    }
    ...
```



push(e) : เพิ่มข้อมูล

- นำ e ไปต่อเป็นใบถัดไป (เพิ่มท้ายในอาเรย์)
- สลับ e กับปมพ่อ จนกว่า e จะไม่มากกว่าพ่อ



© S. Prasitjutrakul 2013

push(e)

```
template <typename T, typename Comp = std::less<T> >
class priority_queue {
  protected:
    ...
    void fixUp(size_t c) {
      T tmp = mData[c];
      while (c > 0) {
        size_t p = c / 2;
        if ( mLess(tmp, mData[p]) ) break;  // tmp < mData[p]
        mData[c] = mData[p];
        c = p;
      }
      mData[c] = tmp;
    }
  public:
    void push(const T& element) {
      if (mSize+1 > mCap) expand(mCap*2);
      mData[mSize] = element;
      mSize++;
      fixUp(mSize-1);
    }
```

priority_queue : less comparator

```
template <typename T, typename Comp = std::less<T> >
class priority_queue {
  protected:
    ...
    void fixUp(size_t c) {
      T tmp = mData[c];
      while (c > 0) {
        size_t p = (c-1) / 2;
        if ( mLess(tmp, mData[p]) ) break;
        mData[c] = mData[p];
        c = p;
      }
      mData[c] = tmp;
    }
```

```
namespace std {
 template <class T> struct less : binary_function <T,T,bool> {
   bool operator() (const T& x, const T& y) const {
     return x < y;
   }
 };
...
```

priority_queue : less comparator

```
typedef bool(*CompFunc)(int, int);
bool myGreater(int a,int b) {
  return a > b;
}
```

```
priority_queue<int, std::less<int>>    pq1;
priority_queue<int, std::greater<int>> pq2;
priority_queue<int, CompFunc>          pq3(myGreater);
priority_queue<int, CompFunc>
  pq4( [](int x, int y){ return x > y; } ); // lambda

for (int i=0; i<10; i++) {
  pq1.push(i);
  pq2.push(i);
  pq3.push(i);
  pq4.push(i);
}
cout << pq1.top() << endl;  // 9
cout << pq2.top() << endl;  // 0
cout << pq3.top() << endl;  // 0
cout << pq4.top() << endl;  // 0
```

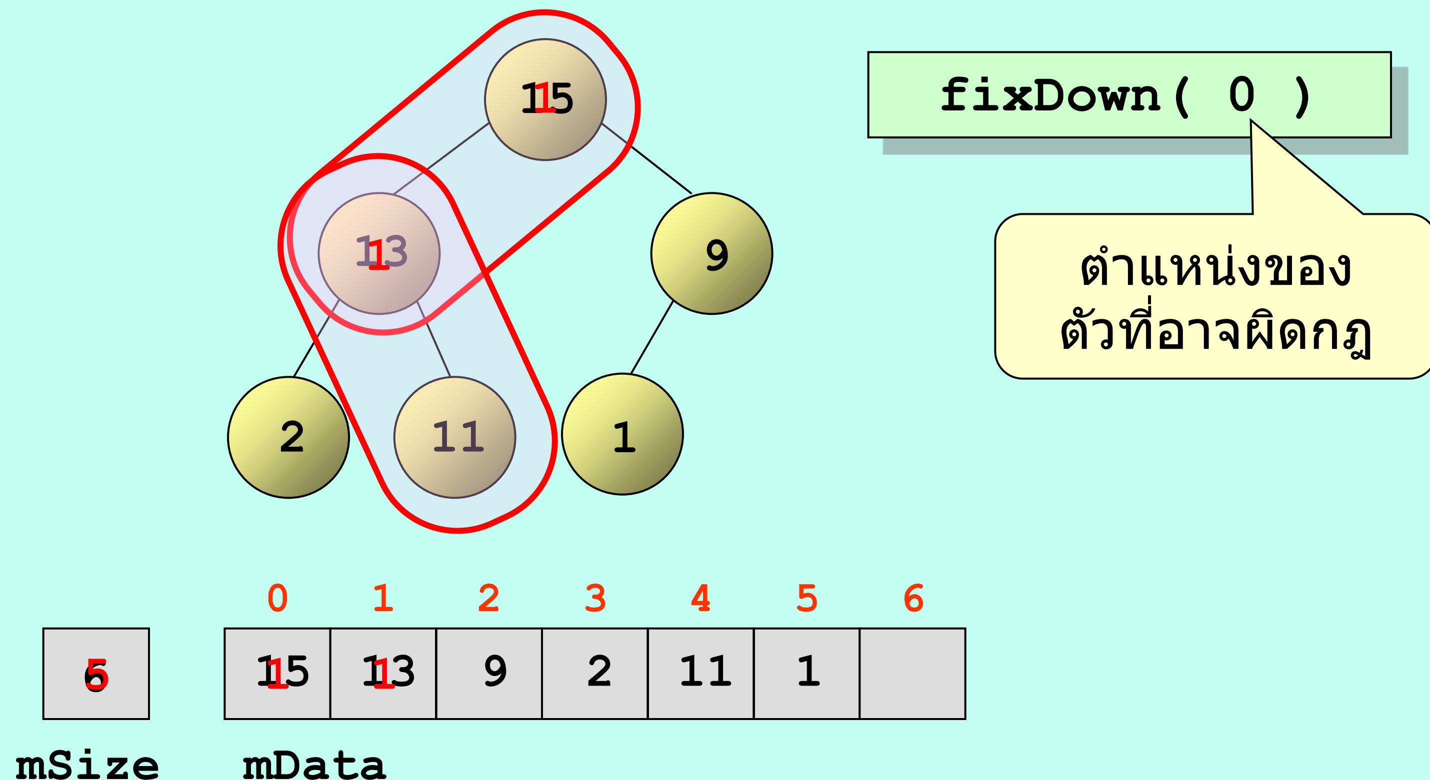
push(e) : อาเรย์เต็มก็ขยาย

```
template <typename T, typename Comp = std::less<T> >
class priority_queue {
  protected:
    ...
    void expand(size_t capacity) {
      T *arr = new T[capacity]();
      for (size_t i = 0;i < mSize;i++) arr[i] = mData[i];
      delete [] mData;
      mData = arr; mCap = capacity;
    }
    ...
  public:
    void push(const T& element) {
      if (mSize+1 > mCap) expand(mCap*2);
      mData[mSize] = element;
      mSize++;
      fixUp(mSize-1);
    }
```

© S. Prasitjutrakul 2013

pop( ) : ลบตัวมากสุด

- เก็บรากไว้เป็นคำตอบ
- ย้ายข้อมูลที่ใบล่างขวาสุด มาเก็บที่ราก
- สลับข้อมูลที่รากลงมา จนกว่าพ่อจะไม่น้อยกว่าลูก



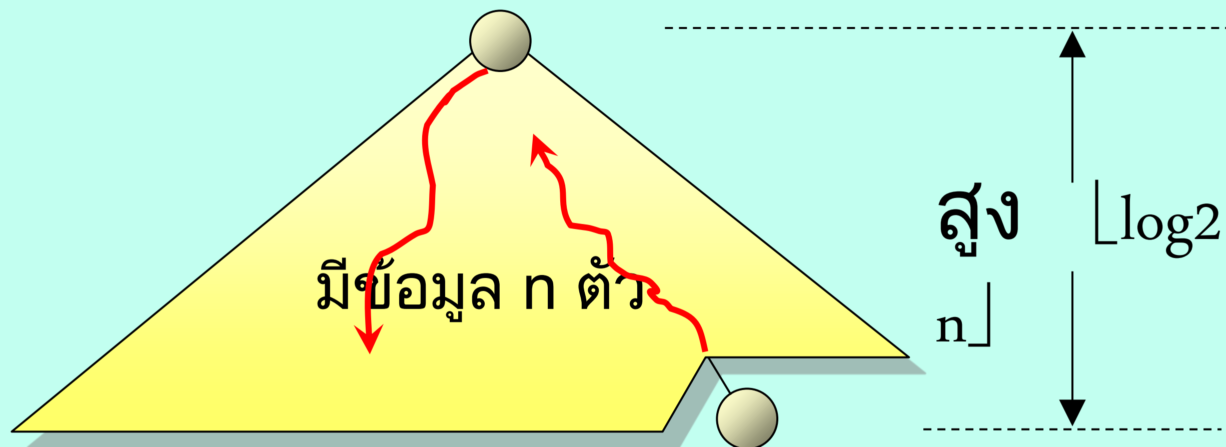
© S. Prasitjutrakul 2013

pop(e) : ลบตัวมากสุด

```
template <typename T, typename Comp = std::less<T> >
class priority_queue {
  protected:
    ...
    void fixDown(size_t p) {
      T tmp = mData[p];
      size_t c;
      while ((c = 2*p + 1) < mSize) {
        if (c+1 < mSize && mLess(mData[c],mData[c+1])) c++;
        if ( mLess(mData[c],tmp) ) break;
        mData[p] = mData[c];
        p = c;
      }
      mData[p] = tmp;
    }
  public:
    void pop() {
      mData[0] = mData[mSize-1];
      mSize--;
      fixDown(0);
    }
```

เวลาการทำงาน

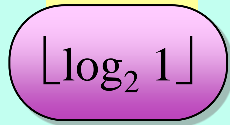
- top : O(1)
- push : fixUp = O(h) = O(log n)
- pop : fixDown = O(h) = O(log n)

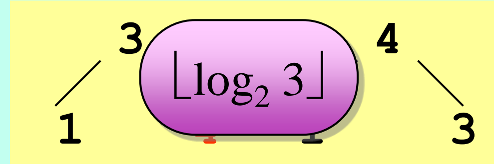


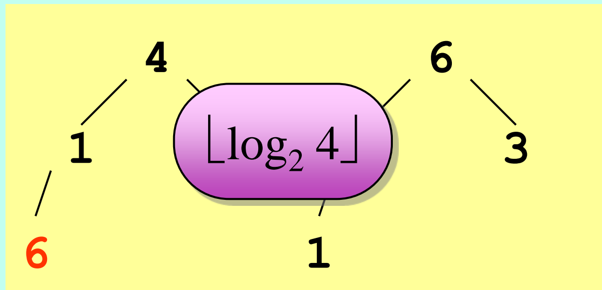
© S. Prasitjutrakul 2013

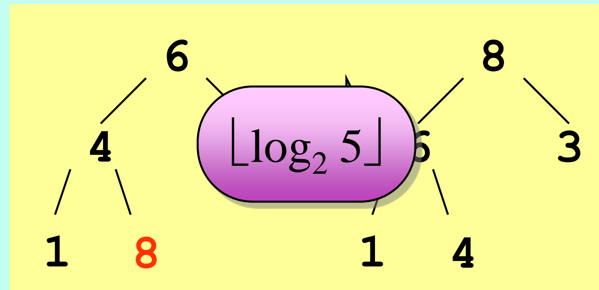
การสร้างฮีปแบบทวิภาคด้วยการค่อย ๆ เพิ่ม

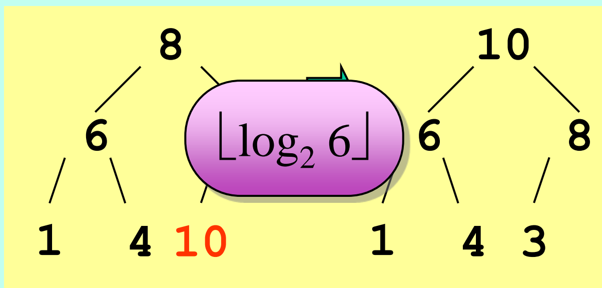
```
priority_queue(T a[], int n, const Comp& c = Comp() ) :
               mData(new T[n]()), mCap(n), mSize(0), mLess(c) {
  for (int i=0; i<n; i++) push(a[i]);
}
```

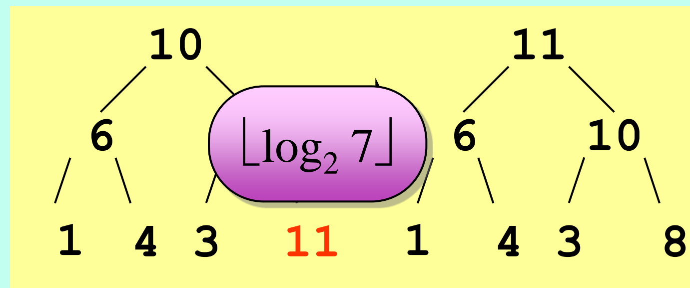












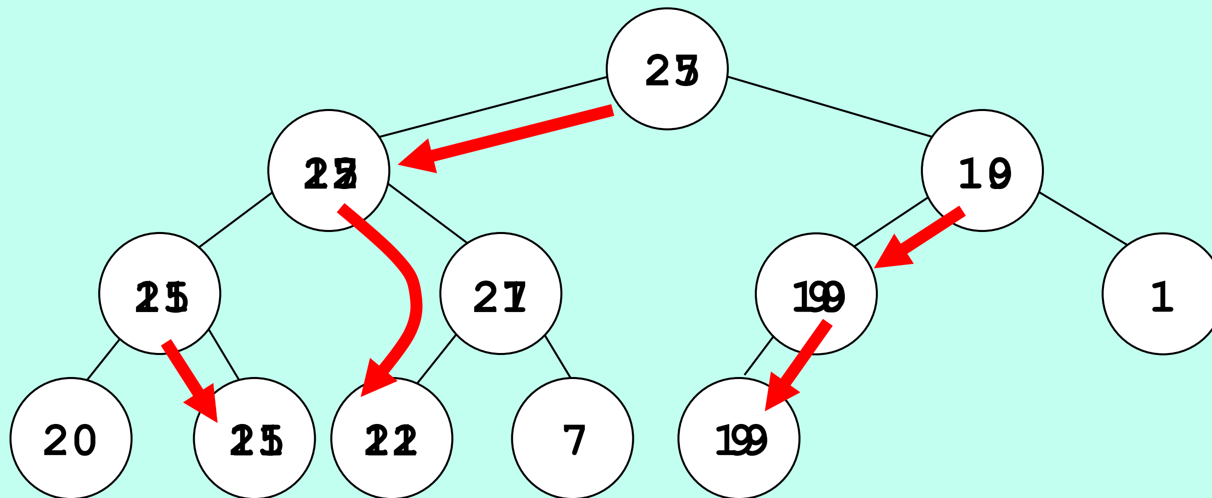
$$\leq \lfloor \log_2 1 \rfloor + \lfloor \log_2 2 \rfloor + \lfloor \log_2 3 \rfloor + \ldots + \lfloor \log_2 n \rfloor < \log_2 n! = O(n \log n)$$

© S. Prasitjutrakul 2013

การสร้างฮีปแบบทวิภาคด้วยการค่อย ๆ ปรับ

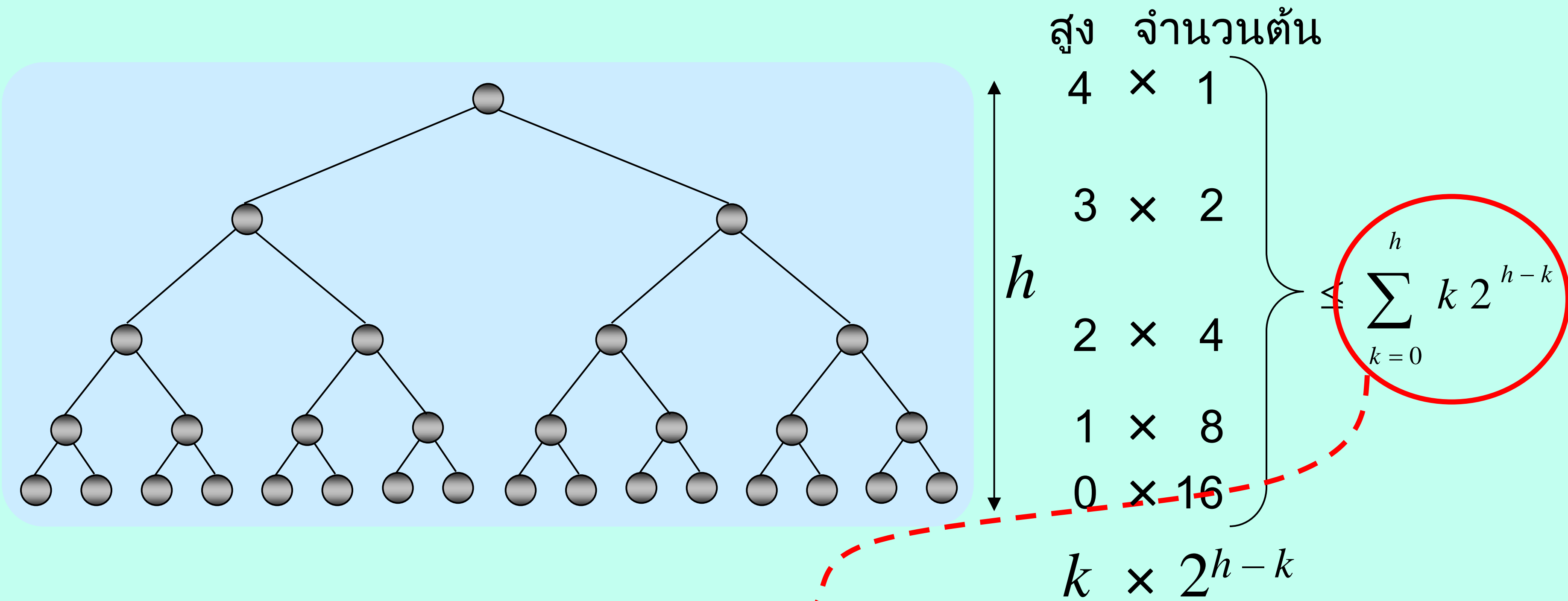
```
priority_queue(T a[], int n, const Comp& c = Comp() ) :
               mData(new T[n]()), mCap(n), mSize(n), mLess(c) {
  for (int i=0; i<n; i++) mData[i] = a[i];
  for (int i=mSize/2-1; i>=0; i--) fixDown(i);
}
```

0	1	2	3	4	5	6	7	8	9	10	11
25	12	10	11	27	9	1	20	25	21	7	19



© S. Prasitjutrakul 2013

การสร้างฮีปแบบทวิภาคด้วยการค่อย ๆ ปรับ



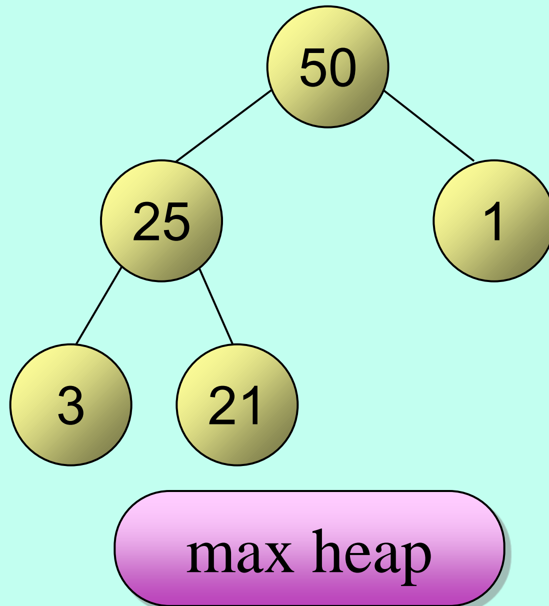
fixdown ในต้นไม้สูง k เกิดการสลับข้อมูลไม่เกิน k ครั้ง

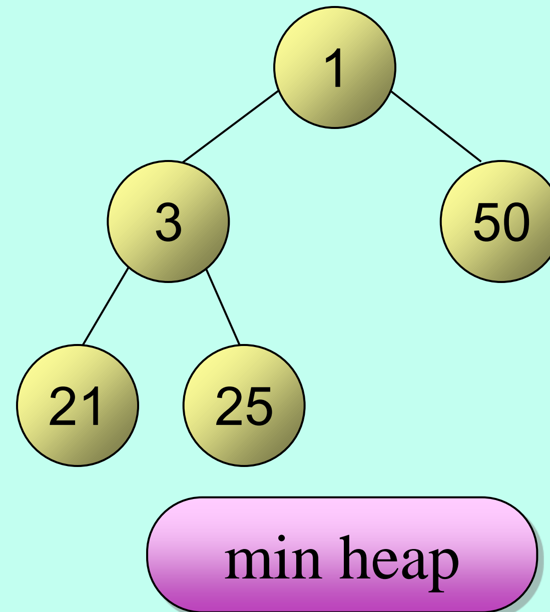
$$\sum_{k=0}^{h} k\, 2^{h-k} = 2^{h} \sum_{k=0}^{h} k\, 2^{-k} < 2^{h} \sum_{k=0}^{\infty} k\, 2^{-k} = 2^{h+1} = O(n)$$

2

ฮีปมากสุด / ฮีปน้อยสุด (Max/Min Heap)

- ฮีปมากสุด
  - ข้อมูลของ parent node มีค่ามากกว่าของลูก ๆ
- ฮีปน้อยสุด
  - ข้อมูลของ parent node มีค่าน้อยกว่าของลูก ๆ





© S. Prasitjutrakul 2013