ダイクストラのアルゴリズムの計算量は、外側のループ(a)が n 回まわり、内側の処理(b, c)に O(n)かかるので、合計で $O(n^2)$である。しかし、辺の数 e が節点の数 n と同程度の場合には、節点からでている辺の集合をリストで管理し、(b)の処理を優先度付待ち行列を用いて効率化することで、全体の計算量を O((e+n) logn)に抑えることができる。この場合に擬似コードは以下のようになる。

```
ダイクストラ(有向グラフ(V,E), 辺のコスト d[ ], 出発点 s){
    for( v in V)    // 初期化。直接移動のコストを C にセットする。
        C[v] ← d[s, v];
    ヒープ S にVのすべての要素 v を加える。順序は C[v]を基準とする。
    while( S が空でない)
        w ← S から C[w]が最小の頂点 w を取り出す
        for ( v ← w から出ている辺の行き先)
            if ( C[w] + d[w, v] < C[v] )
                C[v] ← C[w] + d[w, v]    //コストの更新
                ヒープにおける v の位置を C[v]に従って繰り上げる
                    （逆転がなくなるまで上に上げていく。）
    return C;
}
```

ヒープへの要素の追加、位置の更新・最小値の取り出し、にはそれぞれ O(log n)かかる。最小値を取り出す操作は n 回、位置の更新は合計で最大 e 回実行されるので、全体では O((n+e) log n)となる。一般的に辺の数は頂点の数と同程度のオーダーである疎なグラフであることが多いので、全体の計算量は O(n log n)となり、このアルゴリズムが有効である。しかし、すべての頂点間が結ばれた完全グラフ($e=n^2$)のような場合には $O(n^2 \log n)$となり、逆に効率が悪くなるので注意が必要である。